

CECOM

CENTER FOR SOFTWARE ENGINEERING
ADVANCED SOFTWARE TECHNOLOGY

DTIC
JUN 21 1990
S E
CO

Subject: **Final Report - Real-Time Ada Problem
Solution Study**

CLEARED
FOR OPEN PUBLICATION
SEP 20 1989
DIRECTORATE FOR FREEDOM OF INFORMATION
AND SECURITY REVIEW (OASD-PA)
DEPARTMENT OF DEFENSE

CIN: C02-092LA-0006-00

24 MARCH 1989

894231

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

EXECUTIVE SUMMARY

The purpose of this study was to identify and document solutions to some of the problems that can occur during the development and implementation of real-time embedded Ada applications. The generic Ada problems that are addressed in this report were defined in a previous study.

The study was performed using a three-step approach. During the first phase, theoretical solutions were proposed for the generic Ada problems. These theoretical solutions were based on actual Ada development experience and our understanding of the Ada language and its runtime system (RTS). During the second phase, Ada developers that had current or recent experience in developing Ada projects were interviewed to provide empirical solution data. To complete the study, the theoretical and actual solutions were compared, the similarities and differences between these solutions were analyzed, and this data was used to develop conclusions and recommendations.

As a result of this study, a number of solutions were identified for the generic Ada problems. Some of the solutions are purely theoretical and may or may not be practical. Other solutions are proven, having actually been implemented by Ada developers on their projects. The solutions were categorized according to their characteristics: short-term vs. long-term, preventive vs. remedial, and approach (technical, management, tools, or methodology).

The overall results of the study are:

- 1) Ada compiler and support tool improvements are reducing (but not yet eliminating), the need to use certain alternate methods to improve system performance for Ada applications. These alternate methods include modifications of the Ada compiler and runtime library (RTL) by the Ada developers and compiler vendors, restricting the use of certain Ada features (such as tasking), and extensive system optimization to improve performance.
- 2) Ada developers are willing to modify and/or customize their Ada runtime libraries (RTLs), using either in-house resources or support from compiler vendors, to obtain performance improvements.
- 3) Ada developers are providing comprehensive training to their employees to improve their productivity and ensure that the development team uses the Ada language in an efficient and effective manner. The training includes language usage, methodologies, tool usage, and management issues.

4) Now that more Ada projects are being and have been performed, there is a larger base of information concerning standards and guidelines for Ada usage, as well "lessons learned". A number of developers are taking advantage of this information for use on their own projects.

TABLE OF CONTENTS

1.	INTRODUCTION.....	1
1.1	Purpose.....	1
1.2	Terminology.....	1
1.3	Report Organization.....	1
2.	TECHNICAL DISCUSSION.....	2
2.1	Generic Real-Time Ada Problems.....	3
2.2	Solutions and Avoidance Techniques.....	4
2.3	Technical Findings.....	4
2.3.1	LACK OF KNOWLEDGE CONCERNING ADA RTE.....	13
2.3.2	IMPACT OF ADA COMPILER IMPLEMENTATION DIFFERENCES.....	16
2.3.3	IMPACT OF INTERRUPT HANDLING OVERHEAD.....	19
2.3.4	IMPACT OF MEMORY MANAGEMENT OVERHEAD.....	22
2.3.5	IMPACT OF RUNTIME SYSTEM OVERHEAD.....	24
2.3.6	IMPACT OF TASKING OVERHEAD.....	26
2.3.7	INEFFICIENCY OF OBJECT CODE GENERATED BY ADA COMPILERS.....	30
2.3.8	REQUIREMENTS FOR EXTENSIVE ADA OPTIMIZATION.....	33
2.3.9	INADEQUATE DEBUGGING CAPABILITIES PROVIDED BY CURRENT DEBUGGERS.....	36
2.3.10	ADA EXCEPTION HANDLING.....	38
2.3.11	INEFFICIENCIES IN USING GENERICS.....	40
2.3.12	INABILITY TO PERFORM INDEPENDENTLY OF THE RTS.....	42
2.3.13	LACK OF A DISTRIBUTED RUNTIME LIBRARY (RTL).....	44
2.3.14	INABILITY TO PERFORM SECURE ADA PROCESSING.....	46
2.3.15	DIVERSITY IN IMPLEMENTATION OF APSES.....	48
2.3.16	POOR PERFORMANCE OF ADA TOOLS.....	51
2.3.17	DIFFICULTY IN BENCHMARKING ADA SYSTEMS.....	53
2.3.18	LACK OF ADA SOFTWARE DEVELOPMENT TOOLS.....	56
2.3.19	ADA LANGUAGE COMPLEXITY.....	59
2.3.20	CUSTOMIZATION OF THE RUNTIME LIBRARY.....	61
2.3.21	LACK OF EXPERIENCED ADA PROGRAMMERS.....	63
2.3.22	EXTENSIVE ADA TRAINING REQUIREMENTS.....	67
2.3.23	INACCURACY OF COST/SCHEDULE ESTIMATES FOR ADA PROGRAMS.....	70
2.3.24	LACK OF ESTABLISHED ADA SOFTWARE DEVELOPMENT METHOD.....	74
2.3.25	LACK OF ESTABLISHED ADA STANDARDS AND GUIDELINES.....	77
2.3.26	PRODUCTIVITY IMPACTS OF ADA.....	80
2.3.27	IMPACT OF CONSTRAINT CHECKING ON SYSTEM PERFORMANCE.....	83
2.3.28	INABILITY TO ASSIGN DYNAMIC TASK PRIORITIES.....	85
2.3.29	INABILITY TO PERFORM PARALLEL PROCESSING..	87
2.3.30	LACK OF SUPPORT FOR LOW LEVEL OPERATIONS..	89

2.3.31	INABILITY TO PERFORM TASK RESTART.....	91
2.3.32	INABILITY TO PERFORM CYCLIC SCHEDULING IN ADA.....	93
2.3.33	LACK OF FLOATING POINT COPROCESSOR SUPPORT.....	95
2.3.34	INABILITY TO RECOVER FROM CPU FAULTS IN ADA.....	97
2.3.35	IMPACT OF ADA COMPILER VALIDATION ISSUES..	99
2.3.36	INABILITY TO PERFORM ASYNCHRONOUS TASK....	101
2.3.37	LACK OF IMPLEMENTATION OF CHAPTER 13 FEATURES.....	103
3.	ANALYSIS AND CONCLUSIONS.....	105
3.1	Analysis	105
3.2	Conclusions.....	105
4.	RESULTS AND RECOMMENDATIONS.....	107
4.1	Results.....	107
4.2	Recommendations.....	108
APPENDIX A: DEFINITIONS.....		110
APPENDIX B: REFERENCES.....		113

TABLE OF FIGURES

FIGURE 1:	THEORETICAL SOLUTIONS FOR GENERIC ADA PROBLEMS.....	5
FIGURE 2:	ACTUAL SOLUTIONS FOR GENERIC ADA PROBLEMS.....	9
FIGURE 3:	PROBLEMS SOLUTIONS RESULTS.....	108-1 to 108-6

1. INTRODUCTION

1.1 Purpose

The purpose of this report is to identify and document solutions to some of the problems that can occur during the development and implementation of real-time embedded Ada applications. The problems that are being addressed were identified in previous studies and papers [Soni87, Fran87, Labt87, McCu88]. The solutions provided in this report consist primarily of preventive (avoidance) techniques and direct responses to actual problems.

1.2 Terminology

The definitions given in DoD Standard 2167A [DoD88] and the ANSI IEEE definitions [ANSI83] apply to all terms used in this report. Additional terms are defined in Appendix A.

1.3 Report Organization

Section 1 explains the purpose of this report.

Section 2 discusses the technical approach and the technical findings of the study.

Section 3 contains an analysis of the solutions that were obtained during the study and an analysis of the correlation between the theoretical and actual solutions.

Section 4 presents the results that were obtained during this study and our overall recommendations, based on the study results.

Appendix A contains a definition of terms were used in the report.

Appendix B contains a list of references for other documents that were used as sources of information for the study.

2. TECHNICAL DISCUSSION

A four-step approach was used to perform this study:

- 1) Define theoretical solutions and avoidance techniques that are potentially usable for generic real-time Ada problems.
- 2) Interview participants of actual Ada projects to confirm or disprove the theoretical findings and to possibly add new solutions and avoidance techniques for some of the problems.
- 3) Summarize the findings of the investigation in a matrix of generic real-time Ada problems vs. types of solutions or avoidance techniques.
- 4) Enter the information obtained from the study into the data base that was established to contain information concerning Ada problems and solutions.

As a result of this study, a number of solutions were identified for the generic Ada problems. Some of the solutions are purely theoretical and may or may not be practical. Other solutions are proven, having actually been implemented by Ada developers on their projects.

In some cases, the theoretical and actual solutions for a particular problem are fairly similar. In most cases, this means that the theoretical solutions were pretty accurate; a certain amount of this was expected due to the extensive actual Ada experience of the principal investigators. In those cases where the theoretical and actual solutions were significantly different, this could mean that project-specific circumstances caused the implementation of a solution that may not have been obvious initially or that the proposed theoretical solution(s) were not sufficient. Analysis was performed to determine the meaning of the study results.

The types of solutions identified during the study fell into three categories:

1) Solution Method(s)

Technical - Requires modification or enhancement of the application software design or implementation.

Methodology - Requires modification or enhancement of the software development methodology to reflect the characteristics of the Ada application.

Tools - Requires modification or enhancement of the Ada compiler, runtime library (RTL) or support tools.

Management - Requires modification or enhancement of the project management approach to address Ada issues.

2) Solution Timeframe

Short-term - The solution will provide results in less than one year. Provides a quick fix but may have adverse or unknown long-term impacts.

Long-term - The solution takes longer than one year to provide results. Provides an eventual solution but may have delayed, adverse, or unknown short-term impacts.

3) Solution Approach

Preventive - Used to prevent a problem from occurring.

Remedial - Used to solve a problem once it occurs.

2.1 Generic Real-Time Ada Problems

The Ada programming language is fast becoming a viable real-time programming language for communications, avionics, space and other important areas. Ada was mandated by the military [DoD87] for its real-time projects, but commercial applications are finding Ada the language of choice for many large projects--especially those with over one million lines of code.

Paragraph 2.3 will define the 37 generic real-time Ada problems found in early weapon system applications experience. Not all problems encountered by real-time Ada developers were directly related to the language. In fact, one particular problem showed every symptom of an Ada runtime support library problem--that is, until a system verification by Intel proved otherwise. So, it was not until problems were traced to unique Ada features that they were called Ada problems.

Each Ada problem was considered generic after it had been reported by more than one source. In addition, problems that stemmed from a common cause were grouped together as symptoms of the root problem. For example, problems were identified in using several Ada project support environment tools, but these were grouped together to form a single problem (#16). In this way the original set of hundreds of problems was reduced to a more manageable set of 37.

2.2 Solutions and Avoidance Techniques

There are three ways to overcome an Ada problem which is impacting project development goals:

- 1) Solve the problem within the bounds of accepted software engineering principles.
- 2) Avoid the problem by choosing a development approach within the bounds of accepted software engineering principles.
- 3) Work around the problem, violating accepted software engineering principles in the process.

Because the third approach often causes worse problems than it solves, this study will focus on the first two approaches, problem solutions and problem avoidance techniques.

2.3 Technical Findings

Figure 1 presents a matrix of generic real-time Ada problems versus the types of theoretical solutions or avoidance techniques that address them.

Figure 2 presents a matrix of generic real-time Ada problems vs. the actual solutions obtained from the interviews with Ada developers.

In the subparagraphs below, each generic real-time Ada problem is defined and each theoretical or actual solution or avoidance technique shown by an "X" in Figures 1 and 2 is described.

MATRIX OF PROBLEMS VS. SOLUTIONS 7/25/88

FIGURE 1: THEORETICAL SOLUTIONS FOR GENERIC ADA PROBLEMS

TECHNICAL										
PROB NO.	PROB TITLE	ANALYSIS & EVAL.		BENCHMARK	DESIGN	PROTOTYPE	ADA		ADD'L COMP RESOURCES	
		REVAL.	FEAT.							
1	LACK OF KNOWLEDGE CONCERNING ADA RTE									
2	IMPACT OF ADA COMP INPL DIFFERENCES									
3	IMPACT OF INTRPT HANDLG O/H			H	H	H	H	H		
4	IMPACT OF MEN NGHT O/H			H	H	H	H	H		
5	IMPACT OF RTS O/H			H		H				
6	IMPACT OF TASKING O/H			H		H				
7	INEFF OF OBJ CODE GEN								H	
8	RGMTS FOR EXTENSIVE ADA OPTIM			H		H				
9	INADEQ C.P PROVIDED BY CURR DEBUG									
10	DELAYED HANDLING OF ADA EXCEPTIONS				H(2)					
11	IMPACT OF EXTENSIVE USE OF GENERICS									
12	INABILITY TO PERF INDEP OF RTS									
13	LACK OF A DISTR RTS (MULTIPLE PROC)				H					
14	INABILITY TO PERF SECURE ADA PROC									
15	DIVERSITY IN INPL OF RPSE'S									
16	POOR PERFORMANCE OF ADA TOOLS									
17	DIFF IN BENCHMARKING ADA SYSTEMS		H(2)	H						
18	LACK OF ADA S/H DEVELOPMENT TOOLS									
19	ADA LANGUAGE COMPLEXITY		H							
20	RGMTS FOR CUSTOMIZATION OF RTL									
21	LACK OF EXP ADA PROGRAMMERS									
22	EXTENSIVE ADA TRNG REQUIREMENTS									
23	INACCY OF C/S EST FOR ADA PROG									
24	LACK OF ADA EST SM DEV METHOD									
25	LACK OF EST ADA SM STDS AND GUIDE									
26	PRODUCTIVITY IMPACTS OF ADA									
27	IMPACT OF CONSTRAINT CHK ON SYS. PERF						H	H		
28	INABILITY TO ASSIGN DYN TASK PRIORITIES						H	H		
29	INABILITY TO PERF PARALLEL PROCESSING									
30	LACK OF SUPPT FOR LOW LEVEL OPERATION				H(2)					
31	INABILITY TO PERF TASK RESTART									
32	INABILITY TO PERF CYCLIC SCH IN ADA				H		H			
33	LACK OF FLTING PT COPROCESSOR SUPPORT				H					
34	INABILITY TO RECOVER FROM CPU FAULTS									
35	IMPACT OF ADA ACVC ISSUES									
36	INABILITY TO PERF ASYNC. TASK									
37	LACK OF IMPLEMENTATION OF CHAPTER 13				H					

MATRIX OF PROBLEMS VS. SOLUTIONS
7/25/84

FIGURE 1 (CONT.): THEORETICAL SOLUTIONS FOR GENERIC ADA PROBLEMS

		ADA SOFTWARE TOOLS					ANALYSIS	
PROB NO.	PROB TITLE	MODIFY	EMBRACE	ACQUIRE	BUILD	ADA-ORIENTED	VALIDATE	& EVAL.
1	LACK OF KNOWLEDGE CONCERNING ADA RTE			N				N/C/D
2	IMPACT OF ADA COMP INPL DIFFERENCES							N
3	IMPACT OF INTRPT INADJ O/H	N						N
4	IMPACT OF NEW INTRPT O/H	N						N
5	IMPACT OF RTS O/H							
6	IMPACT OF TASKING O/H	N	N					N
7	INEFF OF OBJ CODE GEN	N						N
8	ADJTS FOR EXTENSIVE ADA OPTIM	N						N
9	INADJQ CMP PROVIDED BY CURR DEBUG			N	N			N
10	DELAYED HANDLING OF ADA EXCEPTIONS							
11	IMPACT OF EXTENSIVE USE OF GENERICS							N
12	INABILITY TO PERF INTRP OF RTS	N						
13	LACK OF A DISTA RTS QUALITATIVE PROC	N						
14	INABILITY TO PERF SECURE ADA PROC	N			N			
15	DIVERSITY IN INPL OF APSE'S	N						N
16	POOR PERFORMANCE OF ADA TOOLS				N			N
17	DIFF IN DENCIMARKING ADA SYSTEMS							
18	LACK OF ADA S/W DEVELOPMENT TOOLS	N	N		N			N
19	ADA LANGUAGE COMPLEXITY							
20	ADJTS FOR CUSTOMIZATION OF RTL	N						
21	LACK OF EXP ADA PROGRAMMERS			N				
22	EXTENSIVE ADA TRNG REQUIREMENTS			N				
23	INACCY OF C/S EST FOR ADA PROG			N		N		
24	LACK OF ADA EST SH DEV METHOD			N				
25	LACK OF EST ADA SU STDS AND GUIDE			N				
26	PRODUCTIVITY IMPACTS OF ADA							
27	IMPACT OF CONSTRAINT CLK ON SYS. PERF	N						
28	INABILITY TO ASSIGN DYN TASK PRIORITIES	N						
29	INABILITY TO PERF PARALLEL PROCESSING			N	N			
30	LACK OF SUPPT FOR LOW LEVEL OPERATION							
31	INABILITY TO PERF TASK RESTART	N						
32	INABILITY TO PERF CYCLIC SCH IN ADA							
33	LACK OF FLTING PT COPROCESSOR SUPPORT	N						
34	INABILITY TO RECOVER FROM CPU FINALS	N						
35	IMPACT OF ADA ACVC ISSUES						N/C/D	N
36	INABILITY TO PERF ASYNC. TASK							
37	LACK OF IMPLEMENTATION OF CHAPTER 13			N				

MATRIX OF PROBLEMS VS. SOLUTIONS
7/25/80

FIGURE 1 (CONT.): THEORETICAL SOLUTIONS FOR GENERIC ADA PROBLEMS

PROB NO.	PROB TITLE	ADA SOFTWARE REVISIONS			
		MODIFY	BUILD	ADA-ORIENTED	STANDARDS & GUIDES
1	LACK OF KNOWLEDGE CONCERNING ADA RTE				
2	IMPACT OF ADA COMP IMPL DIFFERENCES				
3	IMPACT OF INTERT MNUAL O/N				
4	IMPACT OF NEW MGMT O/N				
5	IMPACT OF RTS O/N				
6	IMPACT OF TASKING O/N				
7	INEFF OF OBJ CODE GEN				
8	POINTS FOR EXTENSIVE ADA OPTIM				
9	INDEQ CAP PROVIDED BY CURR DEBAG				
10	DELAYED HANDLING OF ADA EXCEPTIONS				
11	IMPACT OF EXTENSIVE USE OF GENERICS				
12	INABILITY TO PERF INDEP OF RTS				
13	LACK OF A DISTR RTS MULTIPLE PROC				
14	INABILITY TO PERF SECURE ADA PROC				
15	DIVERSITY IN IMPL OF APSE'S				
16	POOR PERFORMANCE OF ADA TOOLS				
17	DIFF IN BENCHMARKING ADA SYSTEMS				
18	LACK OF ADA S/W DEVELOPMENT TOOLS				
19	ADA LANGUAGE COMPLEXITY				
20	POINTS FOR CUSTOMIZATION OF RTL				
21	LACK OF EXP ADA PROGRAMMERS				
22	EXTENSIVE ADA TRNG REQUIREMENTS				
23	IMPACT OF C/S EST FOR ADA PROG				
24	LACK OF ADA EST S/W DEV METHOD				
25	LACK OF EST ADA S/W STDS AND GUIDE				
26	PRODUCTIVITY IMPACTS OF ADA				
27	IMPACT OF CONSTRAINT CLK ON SYS. PERF				
28	INABILITY TO ASSIGN DYN TASK PRIORITIES				
29	INABILITY TO PERF PARALLEL PROCESSING				
30	LACK OF SUPPT FOR LOW LEVEL OPERATION				
31	INABILITY TO PERF TASK RESTART				
32	INABILITY TO PERF CYCLIC SCH IN ADA				
33	LACK OF FLING PT COPROCESSOR SUPPORT				
34	INABILITY TO RECOVER FROM CPU FAULTS				
35	IMPACT OF ADA ACVC ISSUES				
36	INABILITY TO PERF ASYNC. TASK				
37	LACK OF IMPLEMENTATION OF CHAPTER 13				

MATRIX OF PROBLEMS VS. SOLUTIONS
7/26/80

FIGURE 1 (CONT.) THEORETICAL SOLUTIONS FOR GENERIC ADA PROBLEMS

PROB NO.	PROB TITLE	MANAGEMENT			
		TRAINING	PLANNING	VENDOR SUPPORT	CONSULT. SUPPORT PERSONNEL
1	LACK OF KNOWLEDGE CONCERNING ADA RTE				
2	IMPACT OF ADA COMP IMPL DIFFERENCES				
3	IMPACT OF INTAPT HARDWARE				
4	IMPACT OF NEW HCRIT O/H				
5	IMPACT OF HTS O/H				
6	IMPACT OF TASKING O/H				
7	INEFF OF OBJ CODE GEN				
8	RUNTIME FOR EXTENSIVE ADA OPTIM				
9	INADEQ CAP PROVIDED BY CURR DEBUG				
10	DELAYED HANDLING OF ADA EXCEPTIONS				
11	IMPACT OF EXTENSIVE USE OF GENERICS				
12	INABILITY TO PERF INDEP OF RTS				
13	LACK OF A DISTR RTS MULTIPLE PROC				
14	INABILITY TO PERF SECURE ADA PROC				
15	DIVERSITY IN IMPL OF APSE'S				
16	POOR PERFORMANCE OF ADA TOOLS				
17	DIFF IN BENCHMARKING ADA SYSTEMS				
18	LACK OF ADA S/W DEVELOPMENT TOOLS				
19	ADA LANGUAGE COMPLEXITY				
20	HOWMIS FOR CUSTOMIZATION OF RTL				
21	LACK OF EHP ADA PROGRAMMERS				
22	EXTENSIVE ADA TRNG REQUIREMENTS				
23	INACCY OF C/S EST FOR ADA PROG				
24	LACK OF ADA EST SW DEV METHOD				
25	LACK OF EST ADA SW SIDS AND GUIDE				
26	PRODUCTIVITY IMPACTS OF ADA				
27	IMPACT OF CONSTRAINT CLK ON SYS. PERF				
28	INABILITY TO ASSIGN DYN TASK PRIORITIES				
29	INABILITY TO PERF PARALLEL PROCESSING				
30	LACK OF SUPP FOR LOW LEVEL OPERATION				
31	INABILITY TO PERF TASK RESTART				
32	INABILITY TO PERF CYCLIC SCH IN ADA				
33	LACK OF FILTERING PT COPROCESSOR SUPPORT				
34	INABILITY TO RECOVER FROM CPU FAULTS				
35	IMPACT OF ADA ADVC ISSUES				
36	INABILITY TO PERF ASYNC. TASK				
37	LACK OF IMPLEMENTATION OF CHAPTER 13				

MATRIX OF PROBLEMS VS. SOLUTIONS
12/12/88

FIGURE 2: ACTUAL SOLUTIONS FOR GENERIC ADA PROBLEMS

PROB NO.	PROB TITLE	TECHNICAL					ADD'L COMP RESOURCES
		ANALYSIS REVAL.	BENCHMARK	DESIGN	PROTOTYPE	ADA FEATURES	
1	LACK OF KNOWLEDGE CONCERNING ADA RTE	H					
2	IMPACT OF ADA COMP IMPL DIFFERENCES						
3	IMPACT OF INTPT MODLG O/H					H	
4	IMPACT OF MEN MGMT O/H						
5	IMPACT OF RTS O/H					H	
6	IMPACT OF TASKING O/H	H					
7	INEFF OF OBJ CODE GEN						
8	ROBNTS FOR EXTENSIVE ADA OPTIM					H	H
9	INREQ CAP PROVIDED BY CURA DEBUG						
10	DELAYED HANDLING OF ADA EXCEPTIONS					H	
11	IMPACT OF EXTENSIVE USE OF GENERICS						
12	INABILITY TO PERF INDEP OF RTS						
13	LACK OF A DISTR RTS (MULTIPLE PROC)						
14	INABILITY TO PERF SECURE ADA PROC						
15	DIVERSITY IN IMPL OF RPSE'S						
16	POOR PERFORMANCE OF ADA TOOLS	H(2)					
17	DIFF IN BENCHMARKING ADA SYSTEMS						
18	LACK OF ADA S/W DEVELOPMENT TOOLS						
19	ADA LANGUAGE COMPLEXITY						
20	ROBNTS FOR CUSTOMIZATION OF RTL						
21	LACK OF EMP ADA PROGRAMMERS						
22	EXTENSIVE ADA TRNG REQUIREMENTS						
23	INACCY OF C/S EST FOR ADA PROG						
24	LACK OF ADA EST SW DEV METHOD						
25	LACK OF EST ADA SW STDS AND GUIDE						
26	PRODUCTIVITY IMPACTS OF ADA						
27	IMPACT OF CONSTRAINT CHK ON SYS. PERF						
28	INABILITY TO ASSIGN DYN TASK PRIORITIES						
29	INABILITY TO PERF PARALLEL PROCESSING						
30	LACK OF SUPPT FOR LOW LEVEL OPERATION						
31	INABILITY TO PERF TASK RESTART						
32	INABILITY TO PERF CYCLIC SCH IN ADA						
33	LACK OF FLTNG PT COPROCESSOR SUPPORT						
34	INABILITY TO RECOVER FROM CPU FAULTS						
35	IMPACT OF ADA ACVC ISSUES						
36	INABILITY TO PERF ASYNC. TASK						
37	LACK OF IMPLEMENTATION OF CHAPTER 13						

MAJORITY OF PROBLEMS VS. SOLUTIONS
12/16/88

FIGURE 2 (CONT.) ACTUAL SOLUTIONS FOR GENERIC ADA PROBLEMS

PROB NO.	PROB TITLE	ADA SOFTWARE TOOLS					ANALYSIS & EVAL
		MODIFY	ENHANCE	ACQUIRE	BUILD	ADA-ORIENTED	
1	LACK OF KNOWLEDGE CONCERNING ADA RTE						
2	IMPACT OF ADA COMP IMPL DIFFERENCES						
3	IMPACT OF INTACT HANDLING O/H	H	H				
4	IMPACT OF NEW INST O/H				H		
5	IMPACT OF RTS O/H	H					
6	IMPACT OF TASKING O/H	H					
7	INEFF OF OBJ CODE GEN	HCD					
8	ADJUSTS FOR EXTENSIVE ADA OPTIM						
9	INSTR CAP PROVIDED BY CURR DEBUG			H			H
10	DELAYED HANDLING OF ADA EXCEPTIONS	H					
11	IMPACT OF EXTENSIVE USE OF GENERICS	H					H
12	INABILITY TO PERF INDEP OF RTS	H	H				
13	LACK OF A DISTA RTS MULTIPLE PROC						
14	INABILITY TO PERF SECURE ADA PROC						
15	DIVERSITY IN IMPL OF RPSE'S			H			H
16	POOR PERFORMANCE OF ADA TOOLS						
17	DIFF IN BENCHMARKING ADA SYSTEMS						
18	LACK OF ADA S/W DEVELOPMENT TOOLS			H	H		
19	ADA LANGUAGE COMPLEXITY						
20	ADJUSTS FOR CUSTOMIZATION OF RTL	HCD					
21	LACK OF EXP ADA PROGRAMMERS						
22	EXTENSIVE ADA TRNG REQUIREMENTS						
23	INACCY OF C/S EST FOR ADA PROG			H		H	
24	LACK OF ADA EST S/W DEV METHOD						
25	LACK OF EST ADA S/W DEV GUIDE						
26	PRODUCTIVITY IMPACTS OF ADA			H			
27	IMPACT OF CONSTRAINT CLK ON SYS. PERF						
28	INABILITY TO ASSIGN WPN TASK PRIORITIES	H		H			
29	INABILITY TO PERF PARALLEL PROCESSING				H		
30	LACK OF SUPPT FOR LOW LEVEL OPERATION	H					
31	INABILITY TO PERF TASK RESTART						
32	INABILITY TO PERF CYCLIC SCH IN ADA			H			
33	LACK OF FLTING PT COPROCESSOR SUPPORT			H			
34	INABILITY TO RECOVER FROM CPU FAULTS			H			
35	IMPACT OF ADA ACVC ISSUES						H
36	INABILITY TO PERF ASYNC. TASK			H			
37	LACK OF IMPLEMENTATION OF CHAPTER 13			H			H

MAINTAIN OF PROBLEMS VS. SOLUTIONS
12/12/68

FIGURE 2 (CONT.) : ACTUAL SOLUTIONS FOR GENERIC ADA PROBLEMS

		ADA SOFTWARE METHODOLOGY			
PROB NO.	PROB TITLE	MODIFY	BUILD	ADA-ORIENTED	STANDARDS & GUIDES.
1	LACK OF KNOWLEDGE CONCERNING ADA RTE				
2	IMPACT OF ADA COMP IMPL DIFFERENCES				
3	IMPACT OF INTRPT MODLG O/N				
4	IMPACT OF NEW MCAT O/N				N
5	IMPACT OF RTS O/N				
6	IMPACT OF TASKING O/N				N
7	IMLFF OF OBJ CODE GEN				N
8	ADJUSTS FOR EXTENSIVE ADA OPTIM				
9	INDEQ CAP PROVIDED BY CURR DEBUG				
10	DELAYED HANDLING OF ADA EXCEPTIONS				
11	IMPACT OF EXTENSIVE USE OF GENERICS				N
12	INABILITY TO PERF INDEP OF RTS				
13	LACK OF A DISTR RTS (MULTIPLE PROC)				
14	INABILITY TO PERF SECURE ADA PROC				N
15	DIVERSITY IN IMPL OF RPSE'S				N
16	POOR PERFORMANCE OF ADA TOOLS				
17	DIFF IN BENCHMARKING ADA SYSTEMS				
18	LACK OF ADA S/N DEVELOPMENT TOOLS				
19	ADA LANGUAGE COMPLEXITY				
20	ADJUSTS FOR CUSTOMIZATION OF RTL				
21	LACK OF EXP ADA PROGRAMMERS				
22	EXTENSIVE ADA TRNG REQUIREMENTS				
23	IMPACT OF C/S EST FOR ADA PROG				
24	LACK OF ADA EST SM DEV METHOD	N	N		
25	LACK OF EST ADA SM STDS AND GUIDE	N	N		NCD
26	PRODUCTIVITY IMPACTS OF ADA				
27	IMPACT OF CONSTRAINT C/M ON SYS. PERF				
28	INABILITY TO ASSIGN DYN TASK PRIORITIES				
29	INABILITY TO PERF PARALLEL PROCESSING				
30	LACK OF SUPPT FOR LOW LEVEL OPERATION				
31	INABILITY TO PERF TASK RESTART				
32	INABILITY TO PERF CYCLIC SCH IN ADA				
33	LACK OF FLTING PT COPROCESSOR SUPPORT				
34	INABILITY TO RECOVER FROM CPU FAULTS				
35	IMPACT OF ADA MISC ISSUES				
36	INABILITY TO PERF ASYNC. TASK				
37	LACK OF IMPLEMENTATION OF CHAPTER 13				

MATRIX OF PROBLEMS VS. SOLUTIONS
12/15/88

FIGURE 2 (CONT.): ACTUAL SOLUTIONS FOR GENERIC ADA PROBLEMS

PROB NO.	PROB TITLE	MANAGEMENT			
		TRAINING	PLANNING	VENDOR SUPPORT	CONSULT. SUPPORT PERSONNEL
1	LACK OF KNOWLEDGE CONCERNING ADA RTE				N
2	IMPACT OF ADA COMP IMPL DIFFERENCES				
3	IMPACT OF INTAPT IMPLD O/H				
4	IMPACT OF NEA NEAT O/H				
5	IMPACT OF RTS O/H				
6	IMPACT OF TASKING O/H				
7	INEFF OF OBJ CODE GEN				
8	ROUTES FOR EXTENSIVE ADA OPTIM				
9	IMDED CAP PROVIDED BY CURA DEMO				
10	DELAYED HANDLING OF ADA EXCEPTIONS				
11	IMPACT OF EXTENSIVE USE OF GENERICS				
12	INABILITY TO PERF INDEP OF RTS				
13	LACK OF A DISTA RTS (MULTIPLE PROC)				
14	INABILITY TO PERF SECURE ADA PROC				
15	DIVERSITY IN IMPL OF APSE'S				
16	POOR PERFORMANCE OF ADA TOOLS				
17	DIFF IN BENCHMARKING ADA SYSTEMS				
18	LACK OF ADA S/M DEVELOPMENT TOOLS				
19	ADA LANGUAGE COMPLEXITY				
20	ROUTES FOR CUSTOMIZATION OF RTL				N(2)
21	LACK OF EXP ADA PROGRAMMERS				N(3)
22	EXTENSIVE ADA TMBG REQUIREMENTS				N
23	INACCY OF C/S EST FOR ADA PROG				N
24	LACK OF ADA EST S/M DEV METHOD				
25	LACK OF EST ADA S/M STDS AND GUIDE				
26	PRODUCTIVITY IMPACTS OF ADA				
27	IMPACT OF CONSTRAINT CHG ON SYS. PERF				
28	INABILITY TO ASSIGN DYN TASK PRIORITIES				
29	INABILITY TO PERF PARALLEL PROCESSING				
30	LACK OF SUPPT FOR LOW LEVEL OPERATION				
31	INABILITY TO PERF TASK RESTART				
32	INABILITY TO PERF CYCLIC SCH IN ADA				
33	LACK OF FLTING PT COPROCESSOR SUPPORT				
34	INABILITY TO RECOVER FROM CPU FAULTS				
35	IMPACT OF ADA ADVC ISSUES				
36	INABILITY TO PERF ASYNC. TASK				
37	LACK OF IMPLEMENTATION OF CHAPTER 13				

2.3.1 LACK OF KNOWLEDGE CONCERNING ADA RTE

2.3.1.1 Problem Definition

The Ada language provides a very complex, powerful, and sophisticated runtime system to support such Ada features as memory management, process control, and others. The primary element in the Ada-supplied runtime system is the Run-Time Library (RTL). The RTL performs a number of activities which include:

- * Memory Management
 - Dynamic memory allocation/deallocation
 - Garbage collection
- * Process Scheduling
 - Task activation/deactivation
 - Task scheduling
 - Task rendezvous
- * Resource Control
 - Resource scheduling
 - Resource monitoring
- * Error Processing (exception handling)

The RTL resides in the target environment during system operation and operates in conjunction with the applications software.

The performance of the RTL affects the performance of the applications programs (tasking, memory management, interrupts, etc.). The RTL is also part of the system debug/testing process since it resides in the target environment. The RTL may require customization as part of the system optimization process. The RTL may need to be benchmarked to obtain an accurate estimate of system performance.

The information concerning the RTL characteristics is vital to the applications developers, since they must address the impact of the RTL during system design and implementation. If this information is not made available to the applications programmers, there is little hope that the system being developed will perform as expected. To date, the Ada compiler vendors have provided little information to the Ada applications developers concerning the detailed sizing, timing, performance, and functional characteristics of the RTL.

2.3.1.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Obtain consulting support from the compiler vendor or another source, such as a consultant or a company that has experience with the selected Ada runtime system (RTS). The vendor support can consist of Ada RTS training, providing "on-call" support to answer specific questions, and reviewing the developer's design and implementation to evaluate its interaction with the RTS. This support can provide the development team with real-world information concerning the capabilities and performances of the runtime system.

Solution Method - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Acquire runtime library (RTL) documentation from the compiler vendor or another source. The compiler vendor can supply documentation which includes functional and design specifications for the runtime software, relevant benchmark and performance data, and source code (if available). Other Ada developers can provide actual benchmark and performance data from similar projects.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Allocate project resources to become familiar with the RTS. This solution involves the creation of a project team that is tasked to become intimately familiar with the selected Ada runtime. This team of "gurus" provides formation concerning the functionality and timing of the RTS to the development team for use in the design and implementation efforts. This solution can also require the allocation of project computing resources for use in performing RTS benchmarks.

Solution Method - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.1.3 Actual Industry Solutions

Solution 1 - Develop a strong knowledge of the Ada RTS. One interviewee stated that his company had gained an in-depth knowledge of the Ada runtime environment because they develop Ada runtime systems as one of their products. This knowledge was particularly useful during the debugging phase, since the RTL code is interleaved with the user code, and any attempt to trace program execution leads the developer into the RTL code.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Obtain the source code and documentation for the RTL from the compiler vendor. One interviewee stated that his company had obtained its knowledge of the runtime environment by reviewing the source code and documentation. This allowed them to more accurately assess the functional and performance impact of the RTS on the application.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Develop a team of Ada "gurus" that have intimate knowledge of the selected RTS. One interviewee stated that the personnel selected for their team of Ada "gurus" had experience with Ada and assembly language, reflecting the fact that their RTL was written using a combination of these two languages. These personnel also worked closely with the compiler vendor to ensure that their information was accurate.

Solution Method - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.2 IMPACT OF ADA COMPILER IMPLEMENTATION DIFFERENCES

2.3.2.1 Problem Definition

The differences in Ada compiler implementation can impact the performance of an Ada application. The ACVC tests that are used by the AJPO to validate candidate Ada compilers primarily verify the functional and syntactical aspects of the compilers, they do not address performance and implementation issues. Between this and the optional implementation issues discussed in Chapter 13 of the Ada Language Reference Manual, a compiler vendor has some flexibility in determining a compiler implementation approach. The areas in which the impacts of compiler implementation differences are most likely to be observed are:

- * The efficiency of the generated Ada object code
- * The size of the RTL
- * The runtime overhead associated with the RTL
- * Implementation of language features (tasking, generics, etc.)
- * Compilation speed (lines of code per minute)

Differences in compiler implementation can have an effect on system performance. They can cause a reduction in system efficiency, thus requiring additional system resources (hardware and software). Impacts can also be felt in the areas of system maintainability (for both the compiler and the applications software) and in modifications that must be made to other Ada support tools (such as the debugger) to reflect the compiler implementation differences.

2.3.2.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Minimize dependence of the design on compiler implementation details. The Ada software should be designed so that system functionality and performance depend as little as possible on the implementation details of the compiler and Ada RTS. These details include data representation formats, implementation of Ada constructs, and utilization of the underlying hardware architecture. This solution improves the maintainability and portability of the resulting software.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Evaluate the differences between compiler implementation features as part of the compiler selection process. The implementation details of the candidate Ada compilers should be analyzed and evaluated to determine their potential impact, if any, on system performance. The details that should be evaluated include the implementation of tasking (probably the most critical), interrupt handling, memory management, generics, and other features. The results of this evaluation can be used by the developers to select a compiler that is best suited for their project.

Solution Method - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 3 - Select alternate compilers that have similar implementations. If the compiler that is originally selected for use on a project does not meet specified requirements, an alternate compiler may have to be selected. The use of a replacement compiler that is similar to the original compiler implementation can minimize the changes that must be made to the Ada application.

Solution Method - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.2.3 Actual Industry Solutions

Solution 1 - Use an Ada compiler for software development that is similar to that used for software implementation. One interviewee stated that he had prototyped critical parts of his application, using an interim compiler, and then switched over to the implementation compiler as it was available. While using the interim compiler, the interviewee used a subset of the Ada language to minimize the amount of throwaway code.

Solution Method - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Attempt to standardize some of the more critical implementation details for current Ada compilers. This standardization could include areas such as interfaces to the RTL, implementation of tasking, and others. A number of the interviewees recommended this solution. This solution is very difficult to implement because of the inability to mandate compiler designs; however, there are efforts currently underway to recommend and implement some of this standardization.

Solution Method	- Tools
Solution Timeframe	- Long-term
Solution Approach	- Preventive

2.3.3 IMPACT OF INTERRUPT HANDLING OVERHEAD

2.3.3.1 Problem Definition

With the embedded system being the primary target for the use of the Ada language, the efficient handling of interrupts becomes a major issue. Interrupts can be defined as hardware or software signals that stop the current processes of the system under specified conditions and in such a way that the processes can be resumed.

In the embedded system environment, interrupts are critical to the ability of the system to respond to real-time events and perform its required functions. Interrupts, in general, signal the occurrence of some predefined event to the embedded system. The embedded system must then perform the correct functions in some determined amount of time. Therefore, any overhead time, time spent on processes that are over and above the required processes, will degrade the ability of the embedded system to meet its functional requirements. To better illustrate the problem, please note the following:

|<----- B ----->|<----- C ----->|
A

A : Represents the occurrence of a interrupt.

B : Represents the overhead time required to stop the current process and switch to the interrupt handler process.

C : Represents the time spent performing the required processes in response to the interrupt.

The problem that exists with Ada today is the overhead time "B" that is required in a Ada program to stop and switch to the interrupt handling process is too large and in many cases unacceptable for embedded system applications. Currently, Ada programs have overhead times in the hundreds of microseconds to milliseconds. The non-Ada embedded system application overhead times have been in tens of microseconds or less.

2.3.3.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Use "fast" interrupts with limited context switching, when feasible. For a number of the current Ada compilers, the time required to service interrupts can be improved through the use of the "fast" interrupt feature, which is an interrupt which performs limited context switching. If the use of this feature is still not enough to provide adequate performance, the developer can request that the compiler vendor further streamline the Ada interrupt handler. The developer can also write a project-specific interrupt handler which replaces the one supplied in the Ada runtime environment, although this can be a difficult task.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Perform interrupt benchmarking and early prototyping to provide performance information to system developers. Early identification of potential performance problems due to Ada interrupt processing would allow the development team to incorporate this overhead into the system design.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Obtain interrupt performance information for use in selecting a compiler. The Ada compiler that is selected for use on the project should be evaluated to determine whether the interrupt handling performance is sufficient for the intended application. This can be done via simulations and benchmarking, once the compiler is available.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 4 - Minimize the amount of interrupt processing that is performed. In some cases, system overhead can be reduced by restricting the use of interrupts to those situations where they are absolutely necessary. System overhead can also be reduced by minimizing the processing that is performed during interrupt servicing.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.3.3 Actual Industry Solutions

Solution 1 - Modify the RTL to reduce the overhead associated with handling interrupts through the use of tasks. One interviewee stated that he modified the RTL to handle interrupts via a user-developed interrupt service routine instead of an Ada task, thus reducing the overhead due to context switching. However, this should be done carefully because streamlining the Ada interrupt handling process results in a loss of functionality, which could impact other areas of the application.

Another interviewee stated that the compiler vendor had provided a mechanism to reduce interrupt handling overhead by using interrupt vectors rather than interrupt queues (via tasks).

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Use a compiler which provides "fast" interrupt processing. A number of interviewees stated that they were able to obtain compilers which provided this feature through the use of compiler pragmas.

Solution Method	- Technical
Solution Timeframe	- Short-term
Solution Approach	- Preventive

2.3.4 IMPACT OF MEMORY MANAGEMENT OVERHEAD

2.3.4.1 Problem Definition

The runtime support provided to Ada applications programs by the Ada RTL includes a number of memory management functions. The primary functions are memory allocation and deallocation and heap storage management (garbage collection). These functions are performed by the RTL either on an as needed basis (memory allocation/deallocation during a context switch) or as required by the application (use of access types to create objects at runtime).

However, the RTL memory management features add a significant amount of runtime overhead to the performance of an Ada system. Since these functions are resident in the target machine and operate in conjunction with the application programs, they also require system resources (CPU, memory). The utilization of system resources by the RTL must be addressed when performing overall system sizing and timing analyses in the applications environment. It is important to know whether the overhead associated with the RTL memory management features is large enough to affect the ability of the system to meet specified requirements.

2.3.4.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Remove or modify the RTL memory management software, if feasible. If the Ada application does not require memory management, removal or modification of this software could result in reduced storage and execution time requirements. The removal of this code may be done by the compiler vendor or possibly through the use of a pragma, if the compiler supports it. It should be noted, however, that since the RTS also makes use of the memory management software, caution should be exercised in removing or modifying this software.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Perform benchmarking and early prototyping to provide performance information about the memory management software to system developers. The performance of the memory management software should be analyzed early in the project life cycle, and this information should be made available to the design team for use in determining sizing and timing requirements for the application.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Obtain memory management performance information for use in compiler selection. The issue of memory management overhead should be addressed during the compiler selection process to ensure that the chosen compiler provides the minimum amount of overhead to the Ada application.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.4.3 Actual Industry Solutions

Solution 1 - Provide guidance for the use of Ada constructs which adversely impact memory management overhead. One interviewee stated that his project did not use Ada constructs such as unconstrained arrays which can have a severe impact on RAM and ROM storage requirements. He also addressed this problem by initially allocating a specific amount of storage for all tasks used (a feature of the compiler) and keeping these tasks alive throughout the processing.

Solution Method - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Perform your own memory management. One interviewee stated that he had performed all dynamic allocation functions at initialization time and after that, all allocation is performed by the Ada application, not the runtime software. However, this can result in an application that is less maintainable and portable due to this user-written project-specific software.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.5 IMPACT OF RUNTIME SYSTEM OVERHEAD

2.3.5.1 Problem Definition

The Runtime Library (RTL) is the total package of software required at runtime to support the execution of the object code generated by the compiler for the application program. Functions such as dynamic memory management, system activation/allocation, interrupt processing, input/output operations, co-processor support, and tasking are all performed by the RTL. The basic problem with the RTLs of today is that they are generally too large and too slow for many embedded system applications. In terms of sizing, the problem is more noticeable when the application program is fairly small. In this case, it's very possible that the RTL can be larger than the application program.

As the application programs grow larger, the proportion of memory taken by the RTL become less, thus the impact is not as great. For timing impacts, the amount of overhead experienced will depend upon what features of the language are used. Generally speaking, the more you use the unique features (tasking, dynamic memory, delays, etc.) of Ada, the more overhead that is incurred. This can be attributed in part to the immaturity of existing RTLs. Because some of the Ada features are new to the implementers of the language, the techniques of implementing them efficiently are still emerging.

2.3.5.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Modify RTL software to improve system performance. This solution can include configuring the RTL to remove unneeded code or modifying the existing RTL code to add functionality or improve performance. Some compilers provide pragmas that can be used to configure the RTL to reduce its size and improve its performance. In certain cases, the compiler vendor may be required to modify the RTL for project-specific reasons.

If the source code is available, the Ada developer can use in-house resources to modify the RTL. However, this solution should be considered as a last resort. The complexity of the RTL requires the developer to have an intimate knowledge of the RTL to correctly implement the desired modifications without affecting other areas of the RTL. Also, the project team would then have to assume the responsibility for maintaining the modified RTL.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Perform benchmarking and early prototyping to provide functional and performance information to system developers. The early availability of RTS performance information is critical to the developers in identifying potential sources of functional and performance problems and devising solutions to these problems. It is very important to perform this benchmarking and prototyping using code that is very similar to that of the actual application that is being developed.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Obtain RTS functional and performance information for use in compiler selection. The functional and performance characteristics of the RTS should be considered heavily in selecting a compiler for a real-time embedded application. This can greatly reduce the adverse impact of the RTS on system performance.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.5.3 Actual Industry Solutions

Solution 1 - Use the Ada compiler linking process to eliminate unneeded code from the RTL. One interviewee stated that initially, the applications developers were forced to name the specific runtime routines to be used. Later, the compiler vendor modified the compiler to only include those runtime routines that are physically required by the applications software.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Rewrite portions of the RTS to improve system performance. One interviewee stated that he had rewritten runtime routines that performed operations such as timer control and interrupt vector processing. Rewriting these routines improved overall system performance, but required the developer to maintain the RTL from that point on. Another interviewee stated that the RTL used on his project allowed the developers to modify those routines that were designated as "user-customizable".

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.6 IMPACT OF TASKING OVERHEAD

2.3.6.1 Problem Definition

One of the key features of the Ada language is tasking. Ada tasks are "entities whose executions proceed in parallel. "This feature gives Ada a great advantage over other high-level languages, but not without a price. The cost is in terms of overhead. Tasking overhead affects the efficiency of the system in both sizing and timing.

Whenever a designer decides to utilize tasking in an Ada program, he will automatically incur an additional cost in terms of additional runtime support code, which can be as high as thirty kilobytes. This code is required to perform the various features (entry calls, accepts, selects,..etc.) of Ada tasking at runtime.

Another sizing problem has to do with the stack requirements of tasks. The designer must allocate enough memory for his application to make available the additional stack memory for task control information. Also, any stack memory required for any runtime procedures called to execute a particular feature must be added to the total size of the task stack allocation. The stack allocation requirements are required for each task declared in the designer's application program. Thus, the problem is compounded.

With the use of tasking, today's applications will experience timing overhead impacts due to tasking features like task allocation, task activation/termination, task switching, synchronization and task rendezvous. To determine what kind of overhead would be incurred by using tasking, a study was performed by Hughes Aircraft Company. The study conducted a series of tests using the DEC Ada Compiler (1.2) on a VAX 8600 (VMS 4.2). The following results show the magnitude of task overhead compared to the processing done within the task itself:

Description	Task Overhead (usec)	Normal Proc. (usec)
1. Task activation and termination	1960	178
2. Task created via an allocator	150	14
3. Producer-Consumer (2 task switches)	503	46
4. Producer-Buffer-Consumer	1220	111
5. Producer-Buffer-Transporter-Consumer	1694	154
6. Producer-Transpt-Buffer-Transpt-Consumer	2248	204
7. Relay	906	82
8. Conditional Entry		
- no rendezvous	170	15
- rendezvous	29	3
9. Timed Entry		
- no rendezvous	254	23
- with rendezvous	33	3
10. Selective Wait with Terminate	127	12
11. Exception during a rendezvous	962	87

[Ref] T.M. Burger and K.W. Nielsen, "An Assessment of the Overhead. Associated with Tasking Facilities and Task Paradigms in ADA", Hughes Aircraft Company.

2.3.6.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Remove tasking software if not necessary. If an Ada application will not be using the Ada tasking feature, this software can sometimes be removed from the runtime environment via compiler modifications by the compiler vendor or through the use of pragmas (if provided). This can be a critical decision, because the runtime software which supports tasking is, by far, the largest component of the RTS and provides the most excessive processing overhead.

However, it should be noted that the the tasking software is required for interrupt handling and delay statements as well as for implementing user-defined tasks. Also, removing this code usually requires extensive modifications to the RTL and can place severe limitations on the types of applications that can be run using the modified RTL.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Enhance tasking software to increase performance. When feasible, the runtime software which supports Ada tasking can be modified to improve its performance in areas such as context switching and rendezvous processing. An example of this is the "Fast Tasking" feature that is now provided by some compiler vendors; this feature minimizes (and in some cases, eliminates) context switching during task activation and deactivation. However, this solution should be carefully evaluated to determine the impacts caused by the resulting loss of functionality.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Perform benchmarking and early prototyping of the Ada tasking feature to provide functional and performance information to system developers. The information obtained from these early efforts is critical to the Ada designers in maximizing system performance.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 4 - Obtain tasking performance information for use in compiler selection. The performance and implementation of the Ada tasking feature should be considered very strongly in the selection of an Ada compiler; the use of tasking can have a major impact on the performance of Ada applications.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 5 - Provide guidelines for the use of tasking in applications programs. Currently, one of the most common solutions to the problem of tasking overhead is to minimize or eliminate the use of tasking in Ada applications because the tasking feature is the largest source of inefficiency for most Ada programs. However, if guidelines are provided, the powerful Ada tasking feature can be used more effectively rather than eliminated.

Solution Method - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.6.3 Actual Industry Solutions

Solution 1 - Restrict use of Ada tasking features to reduce runtime overhead and improve system performance. One interviewee stated that they minimized the use of Ada rendezvous because of the overhead associated with them. Another interviewee stated that he improved performance by minimizing the number of parameters that were passed during task rendezvous. Another interviewee stated that he improved performance by not performing any task creation or termination during runtime.

Solution Method - Methodology
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Modify the Ada runtime software to reduce the overhead associated with Ada tasking functions. One interviewee stated that he modified the runtime software by replacing the task connection and context-switching performed by interrupt handling tasks with an interrupt service routine that was developed by the applications programmers.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.7 INEFFICIENCY OF OBJECT CODE GENERATED BY ADA COMPILERS

2.3.7.1 Problem Definition

As with most first generation compilers for new languages, the Ada compilers today are somewhat inefficient because of the complexity of the Ada language. Unfortunately, the lack of efficient compilers directly impacts the development of embedded systems today. Embedded systems typically have very restrictive requirements on sizing, timing and power consumption. Therefore, any inefficiencies in the object code generation will impact the cost and performance of the typical weapon system. Because the compilers are producing more code than required to implement a particular function the compilation time is longer. As a result, it takes developers somewhat longer to complete the coding process. This means schedule and cost impacts.

2.3.7.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Modify the Ada compiler to generate more efficient object code for a particular application. For example, the compiler can be modified to better utilize the underlying system hardware architecture or to produce code that utilizes the capabilities of special-purpose project-specific hardware. These modifications should be performed by the compiler vendor.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Determine object code efficiency requirements for use in compiler selection. The efficiency of the generated object code should be weighed heavily in selecting an Ada compiler for a real-time embedded application. The generation of inefficient code can be a major contributor to poor system performance for Ada applications. Determining the efficiency of the object code for a particular application may involve the use and evaluation of project-specific benchmarks for critical portions of the project.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Preventive

Solution 3 - Perform required system performance optimization. The inefficiency of the object code generated by the Ada compiler can require that extensive optimization be performed to ensure that the system meets specified performance requirements. To perform this optimization, the developer must have a good understanding of how the compiler works; sometimes the developer can obtain support from the compiler vendor to obtain this understanding.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.7.3 Actual Industry Solutions

Solution 1 - Restrict the use of Ada constructs and features which generate inefficient object code. One interviewee stated that he used case statements instead of compound IF statements to improve performance. Another interviewee stated that he minimized the number of parameters that were passed in procedure calls.

Another interviewee stated that he restricted the use of array initialization at declaration time and achieved a significant increase in performance. Another interviewee stated that he restricted the use of runtime constraint checking; he felt that a good optimizing compiler and careful programming can minimize the need for this feature.

Solution Method - Methodology
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Request that compiler vendor modify the compiler to improve performance. One interviewee requested that the compiler vendor use indirect addressing rather than computation to compute array indexes. Another interviewee stated that he asked the compiler vendor to make changes to the compiler in areas such as the interface to assembly language routines.

However, this solution can result in a compiler that is project specific and must be maintained by the development team rather than the compiler vendor.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Evaluate generated Ada object code to determine how much code is actually being generated and comparing this with the expected amounts. One interviewee stated that he used disassembled listings to determine how much code is actually being generated. This helped his development team to pinpoint those areas which were the least efficient. He also noted that this is compiler-dependent and depends on the compiler configuration.

Solution Method	- Technical
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 4 - Modify the RTL. One interviewee stated that he had modified the Ada RTL to improve system performance and meet specified functional requirements. These modifications included patching the runtime software to reflect project-specific hardware configurations and providing special-purpose interrupt service routines.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

2.3.8 REQUIREMENTS FOR EXTENSIVE ADA OPTIMIZATION

2.3.8.1 Problem Definition

Due to the inefficiency of compiler-generated Ada applications code and the excessive overhead associated with the RTL run-time support, extensive optimization is generally required to improve the performance of Ada systems.

The types of system optimization that are performed include:

- * Customizing the RTL for a particular application
- * Reducing RTL overhead (tasking, memory management, interrupts)
- * Adding special-purpose hardware and software
- * Rewriting applications programs (algorithms)
- * Modifying compiler implementation details
- * Using non-Ada software
- * Providing absolute addressing capability
- * Providing for storage of constants in ROM

However, this extensive optimization can adversely affect system performance and project productivity. The addition of special-purpose hardware and software along with the use of project-specific programs can reduce system portability, reliability, maintainability, reusability, and verifiability, and can increase system complexity. Also, the extensive rework that is performed as part of the optimization efforts can decrease overall project productivity.

2.3.8.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Perform benchmarking and early prototyping to provide functional and performance information to system developers. One of the main reasons for the extensive optimization required for real-time embedded Ada applications is the failure of developers to perform adequate benchmarking and early prototyping of the Ada applications software and run time environment. The use of benchmarking and early prototyping can greatly reduce the amount of rework required during the software life cycle.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - The Ada compiler and RTL should be analyzed and evaluated early in the development cycle to identify potential areas for performance problems. The results of this analysis can be used to address these problems early, thus reducing the amount of optimization that must be done in the later stages of the project.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Modify the Ada compiler and RTL to add functionality and improve performance. More detailed information concerning these modifications can be found in the writeups for Problems 3 - 7 (Sections 2.3.3 - 2.3.7).

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.8.3 Actual Industry Solutions

Solution 1 - Use assembly language to improve performance. One interviewee used assembly language to rewrite portions of the application to provide increased performance. The portion of the code that was rewritten included high-speed device drivers, and critical portions of the application that had to reside in a limited amount of ROM. The assembly language routine were written a separate programs and interfaced to the Ada application.

Another interviewee stated that he rewrote some of the memory and time-critical portions of the application using the Ada "in-line" code feature to improve system performance.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 4 - Modify the RTL to improve system performance. One interviewee modified the RTL to perform more efficient task context switching by reducing the number of system functions that are performed during this process. In certain situations, machine registers were not saved and data that was not critical to the task being swapped out was also not saved.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Perform algorithm modification. One interviewee modified his algorithms to be more efficient in the Ada environment by replacing unconstrained arrays with constrained arrays. Another interviewee stated that he performed static allocation whenever possible to allow for better compiler optimization and also performed bit packing to save storage space.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 4 - Use Pragma "Optimize" to improve applications performance. One interviewee stated that he used Pragma "Optimize" with the time and space options to meet optimization requirements for his application. The choice of time or space optimization is dependent on the application.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 5 - Modifying applications programs to reduce use of inefficient Ada features. One interviewee suppressed the use of constraint checking near the end of the development effort. Another interviewee reduced the use of tasking in his application to improve system performance.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.9 INADEQUATE DEBUGGING CAPABILITIES PROVIDED BY CURRENT DEBUGGERS

2.3.9.1 Problem Definition

Poor debugging tools do not give the engineer adequate control and visibility of the program at runtime. This will directly affect the verifiability of the program and the system.

2.3.9.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Supplement inadequate Ada debugging tools with other debugging tools for target hardware. In some cases, when the debugging tools provided in the Ada support environment are inadequate, additional debugging capabilities can be obtained through the use of off-the-shelf tools which are not Ada-oriented but are designed for use in the target hardware environment. These tools can include machine code debuggers and in-circuit emulators and may require some manual support to make them suitable for use in debugging an Ada application.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Determine Ada debugging tool requirements for use in selecting an Ada Programming Support Environment (APSE). The performance and features of the selected debugger are very important to the success of an Ada project; thus, an evaluation of the proposed debugger should be strongly considered in the selection of an APSE. The debugger should be Ada-oriented and should provide adequate Ada traceback capability.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Preventive

Solution 3 - Provide special-purpose hardware and software to support debugging efforts. The Ada development team may have to build application-specific Ada-oriented debugging tools to supplement inadequacies in the debugger provided in the APSE. These custom tools may include items such as a system performance monitor or system status monitor.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

2.3.9.3 Actual Industry Solutions

Solution 1 - Use an off-the-shelf debugger which is designed for use with the target processor, even though it may not provide direct traceability to the Ada source code. One interviewee stated that he used an off-the-shelf debugger which allowed debugging at the assembly language level; he used disassembled compiler listings to provide traceability to the Ada source code. This approach requires that the developers know a great deal about how the Ada code generator works.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Request that the compiler vendor improve the debugger to meet project requirements. One interviewee requested that the compiler vendor provide tools to help the developers find their way through the Ada code in areas such as context switches.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Modify applications code to improve capability to perform debugging. One interviewee stated that he inserted additional breakpoints into the applications programs to assist in debugging through task context switches. Another interviewee stated that he embedded code within the applications program to assist in debugging. This code performed functions such as recording variable values, recording which programs are executed, and providing sample data to the application.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 4 - Perform an evaluation of required debugging capabilities as part of the compiler selection process. One interviewee stated that he attempted to perform this evaluation based on the requirements of his application (real-time processing, heavy use of tasking, lots of interrupts). However, he was limited by the capabilities of the Ada compiler that was provided as Government Furnished Equipment.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.10 ADA EXCEPTION HANDLING

2.3.10.1 Problem Definition

The whole idea of handling unexpected errors at runtime is a very good concept. But having developed and debugged some Ada programs, one begins to sense there is a problem with the handling of exceptions. The problem has to do with the manner in which an exception is reported to the engineer and the lack of information that is conveyed. This is particularly true when the exception that is raised is a non-user-defined exception. Further, if the application program does not require the inclusion of TEXT_IO, and many will not because of its large size, or no capability to display text messages exists in the runtime program, the problem can be compounded.

This is because it is now possible for an exception to be raised and the engineer not know of its existence until it manifests itself as a failed function much later. Now the engineer must search for the source of the exception and then determine why the exception was raised. Depending on the debugging tools available to the engineer and how far the manifestation of the problem is from the real problem, the determination of the problem can be long, frustrating and costly.

2.3.10.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Build Ada exception handlers which collect information about exceptions. Currently, most Ada runtime environments do not provide enough information about the system state at the time an exception occurs to allow the developer to easily determine the cause(s) of the error. To address this problem, the developer can write Ada exception handling routines which collect and save information such as the time of the error, and the system state when the error occurred.

Solution Method	- Technical
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - If feasible, display error information when the exception occurs. During the development phase of a project, error information collected during exception handling can be printed when the error occurs, if an output device is available. This requires the inclusion of the Text_I/O package for displaying output. This solution may not be feasible for embedded applications where system output and or processing time are limited.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.10.3 Actual Industry Solutions

Solution 1 - Minimize the use of exceptions. One interviewee stated that he minimized the impact of Ada exception handling by allowing only the use of basic Ada exceptions unless absolutely necessary and by writing programs which performed a minimal amount of exception processing.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Adjust process priorities to facilitate display of error messages during interrupt processing. One interviewee stated that he had to ensure that the priority level of the I/O driver was higher than the priority level of the interrupt routine so that the error messages would print out when they occur.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.11 INEFFICIENCIES IN USING GENERICS

2.3.11.1 Problem Definition

The use of Ada generics is regarded as a major step towards the development of reusable Ada programs. However, in the current Ada environment, extensive use of Ada generics can adversely impact the performance of Ada systems and the productivity of Ada development efforts. The extensive use of generics can result in a significant increase in system memory requirements. This is due to the general inefficiency of code that must be shared by a variety of programs, the additional code that is required to implement conditional processing within the shared code, and the additional processing required to implement the use of the actual parameters within the generic instantiation.

There are also impacts on productivity. Each time a generic program is changed, all programs which contain an instantiation of the generic must be recompiled. Also, the generic instantiations are essentially compiled as in-line code, which increases overall compile time.

2.3.11.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Provide guidance for the use of generics. The use of generics can have unexpected impacts on project productivity and system efficiency. Each time a generic program is changed, all programs which contain an instantiation of the generic must be recompiled. Compilation of generics as in-line code can increase system compilation time. Also, system memory requirements can be increased due to the relative inefficiency of the shared code and conditional processing that is common to generics.

However, the Ada generic is a powerful feature of the language which can greatly improve productivity and increase the reusability of the developed Ada software. Thus, providing guidance and ensuring effective use of generics is preferable to eliminating its use altogether.

Solution Method	- Methodology
Solution Timeframe	- Short-term
Solution Approach	- Preventive

Solution 2 - Avoid the use of compilers that compile generics as in-line code. For generics of substantial size that are widely-used, this approach generally causes large increases in system compilation time, thus reducing project productivity.

However, it should also be noted that there could be situations where the implementation of generics as separate copies could be useful so that each copy can be optimized individually for the conditions under which it was instantiated.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Preventive

2.3.11.3 Actual Industry Solutions

Solution 1 - Minimize the use of generics. One interviewee stated that he had minimized the use of generics to improve project productivity, which was reduced due to the extensive requirements for code recompilation.

Solution Method	- Methodology
Solution Timeframe	- Short-term
Solution Approach	- Preventive

Solution 2 - Evaluate the approach used by the compiler vendor to implement generics. One interviewee stated that he chose a compiler which implemented generics as shared code to eliminate the potentially expensive code recompilation and storage requirements that are associated with generics that are compiled as "in-line" code.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Preventive

2.3.12 INABILITY TO PERFORM INDEPENDENTLY OF THE RTS

2.3.12.1 Problem Definition

One of the common requirements for the embedded system is the requirement to perform Built In Test (BIT). BIT is the ability of the embedded system to perform a self-test without external equipment and indicate if the system is good or bad. BIT is usually performed by some combination of hardware and software. One of the key functions in performing a self-test is the setting of the system to a known state. A good example of this would be the initialization of RAM (random access memory). When an embedded system is first powered up, the state of RAM is unknown.

Therefore, one of the functions of BIT is to set RAM to a known state and then verify it. The problem occurs when the application is implemented in the Ada language. The run-time support code is designed to take control of the system at power-up and perform system elaboration. When elaboration is completed all task stacks and variables have been allocated. But the state of memory (RAM) is still indeterminate. If BIT were to run after elaboration it would destroy the state set up by the RTL. Therefore, BIT must execute before the RTL. Today, one must modify the runtime support code to perform this type of function, and to do so can be very costly and time consuming because you are modifying someone else's code.

2.3.12.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Modify RTL to allow programs to run independently. Certain Ada applications, such as those which require built-in testing (BIT) before system start-up or require programs to run concurrently with the Ada application but outside of the RTS may require this capability. These modifications should be made by the compiler vendor.

Independent BIT can be performed having the system start-up vector transfer control to the BIT software before providing control to the RTS. Some Ada compilers already provide the capability to run pre-RTS programs; however, in most cases, these programs cannot be written in Ada.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

2.3.12.3 Actual Industry Solutions

Solution 1 - Modify runtime software to allow applications programs to perform independent of RSL. One interviewee stated that he had patched the runtime environment's initialization routine so that a user-supplied initialization routine would be run before Ada runtime startup.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Obtain an Ada compiler which allows the developer to execute user-developed initialization routines before the runtime software is executed. One interviewee stated that the compiler developer provided a switch that allows the developer to execute non-runtime software first. Another interviewee stated that his compiler provides user configurable code and allows the user to write his own system initialization and bootstrap code.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

2.3.13 LACK OF A DISTRIBUTED RUNTIME LIBRARY (RTL)

2.3.13.1 Problem Definition

The term "concurrent processing" is often mentioned in the same breath as Ada. This is because the Ada is designed with concurrent processing as a goal. This goal is met by creation of tasks. However, many of today's implementations of the Ada Language Reference Manual (LRM) are not capable of true parallel (concurrent) processing. This can be attributed to the lack of RTLs that are designed to be distributed across multiple processors. Another reason may be the LRM itself, since it does not address the requirements for distributed Ada processing.

Many of today's embedded system applications have requirements that warrant the design of systems capable of true parallel processing. Where parallel processing is required, the technique that is often implemented is distributed processing. Distributed processing occurs when computer processes are distributed across multiple computer processing units (CPU) to achieve true parallel processing. Today, an embedded system with multiple processors, implemented in the Ada language, must develop Ada programs for each CPU in the system. Thus, the system incurs the cost of additional memory, timing, and hardware to accommodate an RTL for each processor.

2.3.13.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Acquire an Ada RTS which can be distributed among multiple processors. Current Ada RTSs do not support the development of true distributed applications. To run an Ada application on multiple processors, an independent version of the RTS must reside on each processor and there is no facility for communication between these separate versions of the RTS.

Solution Method	- Tools
Solution Timeframe	- Long-term
Solution Approach	- Remedial

Solution 2 - Modify the RTS to perform distributed functions. The modifications include implementing interprocess and interprocessor communication between the RTSs, to provide the capability for tasks to execute on multiple processors, rendezvous with tasks on other processors, and share data on other processors. These modifications can be performed by the compiler vendors or Ada developers. Currently, Ada developers such as LabTek are attempting to implement these systems.

Solution Method - Tools
Solution Timeframe - Long-term
Solution Approach - Remedial

Solution 3 - Perform distributed processing via the applications software. If an Ada application must perform distributed processing, the developer is currently required to implement the data interfaces between the processors as part of the applications program. However, the lack of a distributed RTS makes it rather difficult to implement a process which runs on multiple processors.

Solution Method - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.13.3 Actual Industry Solutions

Solution 1 - Modify RTL to support distributed processing. One interviewee stated that he wrote extensions to the RTL to handle distributed processing operations. Among the functions performed by these routines were the passing of data and process information between processors.

Solution Method - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Request that the compiler developers provide task and process information so that a task on one processor can rendezvous with a task on another processor. One interviewee stated that this solution is not yet feasible for most compiler vendors.

Solution Method - Tools
Solution Timeframe - Long-term
Solution Approach - Remedial

2.3.14 INABILITY TO PERFORM SECURE ADA PROCESSING

2.3.14.1 Problem Definition

Due to the current unavailability of a secure Ada operating system and the excessive overhead associated with the use of a secure Ada kernel to restrict system memory accesses, it is currently difficult to build a secure processing application in Ada.

The Ada language allows the applications programmer to perform runtime, and system level operations which increase the difficulty involved in protecting classified programs and data within the system. These operations include the creation, access, and destruction of objects at runtime, the use of address specifications to access particular memory locations; and the writing and execution of in-line assembly language programs.

Also, the RTL code is not only resident in the target environment, but runs in conjunction with the applications programs. Thus, to fully verify the security of an Ada system, the RTL code must be evaluated and certified as part of the system certification effort. This is difficult because most Ada applications programmers have little knowledge concerning the operation of the RTL.

2.3.14.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Acquire the Ada Secure Operating System (ASOS) when it is available. The ASOS is a version of the Ada runtime environment that has been modified to provide data and program security for Ada applications. ASOS is currently under development and will probably be available for use sometime in the near future.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Use of special-purpose hardware/software to implement security mechanisms. Currently, Ada applications are forced to implement required security mechanisms through the use of methods such as protected memory areas and limited software kernels.

Solution Method	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 3 - Establish guidelines for the design and implementation of secure Ada implementations. Certain Ada features such as access types and the runtime memory management software (garbage collection, allocation and deallocation) cause difficulties in verifying the security of Ada software systems. The verification process can be simplified by developing guidelines for the use of these features.

Solution Method - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.14.3 Actual Industry Solutions

Solution 1 - Reduce non-determinism of Ada applications with regard to security by restricting use of certain Ada features/constructs. One interviewee stated that he isolated critical information by locating program data in a protected area of memory and not performing garbage collection. He also restricted the use of exceptions because they can interrupt normal processing at any time (non-determinism).

Solution Method - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.15 DIVERSITY IN IMPLEMENTATION OF APSEs

2.3.15.1 Problem Definition

The task of developing Ada software for computers integral to weapon systems is a complex one, and would be impossible without good support tools. The set of these tools to be used with an Ada compiler is called an Ada Program Support Environment (APSE), and the APSE has been the subject of much study since the Ada language was specified.

One of the problems recognized very early by both the Army Communications and Electronics Command (CECOM) and by the Air Force was the need for standardized and portable APSEs. The Air Force effort was lost in a funding problem soon after its inception, and the CECOM effort, which resulted in the Ada Language System (ALS), was terminated and made public domain. A number of users tried the ALS and found tool performance below the range of usability.

This left the situation where each compiler vendor has marketed its own version of an APSE, with each requiring training both for users and for the host computer support engineers. This, in turn, has made it far more difficult to transition from one compiler to another during a project, a necessity all too often brought about by other Ada problems [Problem #17 for example]. The extent of this problem depends upon the complexity of the APSE that you now have and the one you are considering acquiring. If one of the APSEs is the ALS, the problem is very severe.

Col. Wm. Whitaker (Ret), commenting on a WIS report at the Washington Ada Symposium, stated that most programmers make do with a very minimal support environment, and that many of the exotic tools described in the literature either do not work or are not widely used (or both). One study [Ref] defines the minimal tool set needed as a screen editor and an interactive debugger. You are indeed fortunate if your APSE contains a good source-level interactive debugger [Problem #9], but many APSEs contain tool sets which are quite complex, requiring training for all project designers and host support people.

[Ref] S.J.Hanson and R.R.Rosinski, "Programmer Perceptions of Productivity and Programming Tools", Communications of the ACM, Feb 85

2.3.15.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Select APSE's that are compatible with the Ada applications. The wide range of available APSE implementations and the potential impact of using the different implementations makes it very important and difficult to select an APSE which meets the needs of the project. The difficulty of this situation is further increased because the choice of an APSE is usually driven by the choice of a compiler. The tools which comprise the APSE should thoroughly be evaluated in selecting software tools for the project.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Select alternate APSE's that have similar implementations. If the APSE selected for a project is deemed to be inadequate part way through the development cycle, a new APSE must be chosen. The impact to the project of choosing a new APSE can be minimized by selecting an alternate APSE which resembles the original APSE as closely as possible. The areas of impact which can be minimized include functionality, performance, learning curve, and ease of use.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.15.3 Actual Industry Solutions

Solution 1 - Try to standardize the Ada software development environment as much as possible. One interviewee stated that he is trying to use compilers and tool sets that are supported in a variety of environments, thus allowing him to use a consistent set of support tools over a number of projects.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Improve the APSE evaluation and selection process. One interviewee recommended that education and training be provided to Ada developers in selecting APSEs for use on their projects. This education and training could include determination of criteria for use in the evaluation of APSEs and providing information on the composition and capabilities of existing APSEs.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 3 - Use tools that are external to the APSE. One interviewee stated that he supported his Ada project by performing some APSE functions using external tools such as debuggers, documentation tools, and others.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.16 POOR PERFORMANCE OF ADA TOOLS

2.3.16.1 Problem Definition

In recent years, a number of software support tools have been developed for use on Ada development efforts. However, due to the fact that most of these tools have been developed for project-specific purposes or as a part of internal R&D efforts, these tools have generally provided poor performance. Some of the tools that have been developed include compilers, linkers, importers, exporters, debuggers, editors, pretty printers, PDL processors, library managers, and library software (mathematical, etc.).

The poor performance of these Ada tools has had an adverse impact in some areas of programmer productivity. The programmers have had to expend significant effort to develop workarounds in those (frequent) cases where the tools have not performed as expected (advertised). The tools vendors have provided poor documentation and inadequate tool support, and a number of the tools have been delivered with bugs in them. This situation should improve as the Ada software development environment continues to mature and more tools users provide feedback to the tool vendors concerning the performance of the tools.

2.3.16.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Perform careful evaluation and selection of Ada tools (benchmarking, etc.). Due to the poor performance of current Ada tools, it is crucial to perform a comprehensive evaluation of the tools to be used. This evaluation should include performance demonstrations and benchmarks which reflect the actual processing to be performed on the project.

Solution Method(s)	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Preventive

Solution 2 - Establish relationships with tool vendors (support, modifications, etc..). To ensure that adequate support is available when problems occur with Ada software tools, the development organization should establish a working relationship with the appropriate tool vendors. This relationship could include support such as tool modifications and consulting to provide information on tool characteristics and performance.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Develop in-house tools as required to support the development of Ada applications. These in-house tools will generally be developed to perform project-specific functions and can only be developed if project or corporate funds are available to pay for them. Examples of these tools are documentation and program control tools.

However, due to cost and schedule concerns associated with development efforts, an attempt should be made to find an off-the-shelf tool before committing to build one in-house.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 4 - Obtain information from tool users concerning actual tool performance. One of the best sources of information concerning the performance and capabilities of Ada tools are tool users. The results obtained from usage of a tool on an actual project can be invaluable, particularly if it is similar to the current project.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.16.3 Actual Industry Solutions

Solution 1 - Add more computing resources. One interviewee stated that his approach to recovering from the poor performance of the Ada tools was to add additional processors to his system configuration as required.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.17 DIFFICULTY IN BENCHMARKING ADA SYSTEMS

2.3.17.1 Problem Definition

Benchmarks can be of two main types, those that are used to time and size portions of application code (usually using breadboard hardware) and those that are used to evaluate compilers and other support tools. It is the benchmark software for support tools that is addressed in this problem.

Ada is a very powerful, but also very new, programming language for embedded, mission-critical software. Whenever an application is being planned that is significantly different from previous experience in the software organization performing the task, benchmarking is normally relied upon to scope out the job. Unfortunately, the very constructs that make Ada a valuable embedded software language are hard to find in widely available benchmarks.

The benchmark most often cited by compiler vendors seems to be the Dhrystone, which has none of the new Ada constructs (tasking, interrupt handling, direct addressing, etc.) which make it superior to languages like Pascal for embedded systems. In comparison to real application code, the Dhrystone benchmark is too optimistic in both compilation speed (lines of code per minute) and in object code size (bytes per line of source code).

Errors could be a problem if they are not discovered until application code begins to emerge somewhere around the end of the Detail Design Phase. The project is supposed to be ready to code very heavily at that point, yet will find itself with undersized computer resources, both for the host and the target, if the Dhrystone numbers had been believed.

One approach taken to deal with this problem was reported by M. Kamrad of Honeywell at the Jan 87 SIGAda Conference. He recommended postponing the decision of how many processors to put in the system and what software functions run in each until the coding has matured enough that its final size and timing requirements are known. According to D. L. Doty [Ref], this can be a long wait. He has observed size-estimate errors greater than 100 percent at the RFP stage, 75 percent up to the Preliminary Design Review, and 50 percent up to the Critical Design Review.

But all this leaves the compiler vendor in a quandary if he is trying to introduce a new compiler. Unless he can test it against a real application in Ada which is already running under another Ada compiler, it will be difficult to convince potential customers that this new compiler can really handle an embedded Ada application.

[Ref] D.L.Doty, P.J.Nelson, and K.R.Stewart, "Software Cost Estimation Study: Guidelines for Improved Software Cost Estimating" (Vol 2), Final Technical Report by Doty Associates, Inc., for Rome Air Development Center (RADC-TR-77-220), Griffiss Air Force Base, NY, Aug 77, 145pp as cited by: W. Myers, "A Statistical Approach to Scheduling Software Development", IEEE Computer, Dec 78

2.3.17.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Perform early identification of candidate functions for benchmarking. This early identification of these functions can ensure that the benchmark results are available to the design team quickly and also provides the project management personnel with an up-front estimate of the effort required to perform the benchmarking. The candidate functions should include mission-critical functions, uncertain or risky processing requirements, and performance-critical system software.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Develop and perform relevant project-specific benchmarks. It is very important to develop benchmarks which are relevant to the project being developed. This can include the actual algorithms and processing scenarios that will be used on the project as well as the Ada system software (compiler, runtime environment, etc..). Benchmark data that is provided by vendors must be thoroughly analyzed and evaluated to determine its relevance to the current project.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Obtain benchmark information from similar projects. Benchmark data that is obtained from the actual results of similar projects can be very useful in determining project sizing and timing requirements. In some cases, this data may have to be extrapolated to determine its applicability to a specific project.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.17.3 Actual Industry Solutions

Solution 1 - Evaluate available benchmark information on Ada tools. One interviewee stated that he analyzed benchmark information from sources such as the compiler vendor and PIWG to determine whether the selected compiler would meet performance requirements.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Prototype critical applications on target hardware/system. One interviewee stated that he built a prototype on his target system and then performed benchmarks to evaluate system performance. He stated that it was critical to perform these benchmarks in their selected target environment to provide real-world design information and feedback to the Ada development team.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Use reusable proven system components. One interviewee stated that he used reusable software components and off-the-shelf hardware components whenever possible because of the availability of benchmark and performance information.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 4 - Obtain some information from ACEC tests. One interviewee stated that when available, the ACEC tests will provide information concerning the performance of an Ada compiler in certain scenarios.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 5 - Write benchmark software to support system performance analysis. One interviewee stated that he wrote special-purpose performance monitoring and analysis software to assist in the system benchmarking process.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.18 LACK OF ADA SOFTWARE DEVELOPMENT TOOLS

2.3.18.1 Problem Definition

Although a number of software tools have been developed for use in Ada environments, there is still a variety of tools that are desirable to further improve the productivity and performance associated with Ada development efforts. Some of these tools are currently in various stages of development (planning, design, implementation, testing). Some of the tools that would be useful for Ada efforts include:

- * Ada Design Generators
- * Ada Code Generators
- * Ada Source Code Analyzers
- * Ada-Oriented Debuggers
- * Ada Syntax-Directed Editors
- * Ada Cost Estimation Models
- * Ada Project Management Tools
- * Secure Ada Operating Systems
- * Automated Ada Test Tools

The lack of Ada tools affects the overall productivity of development efforts. The availability of these tools would decrease the number of development activities that are currently performed manually. It would also reduce the overall development effort by reducing the requirement for applications programmers to develop their own project-specific tools. The availability would also improve the consistency of the products developed on Ada projects and would help impose and enforce project methodologies and development standards.

2.3.18.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Obtain information concerning Ada tools which exist or are being developed. It is very important to be well-informed concerning the capabilities of existing Ada tools and those which are currently under development - "Know what is out there". This information is critical in the selection of Ada tools for a project.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Modify/enhance existing Ada tools when feasible. In some cases, the modification or enhancement of an existing Ada tool may provide a faster or cheaper solution to a problem than the purchase or development of a new tool. These decisions should be made as early as possible in the project to ensure that the tools are available when needed.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Build new Ada tools when necessary. An application-specific need can sometimes require that a software tool be developed in-house for use on a project. But, due to the risk involved in the development of new tools, and particularly in the still immature Ada environment, these tools should be acquired off-the-shelf, if at all possible.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 4 - Supplement available Ada tools with non-Ada tools for host and target hardware. The use of non-Ada tools to perform some of the development activities on an Ada project can help to compensate for the lack of available Ada tools. Guidance should be provided to instruct the project personnel in the proper application of these tools to an Ada development effort.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.18.3 Actual Industry Solutions

Solution 1 - Develop tools as required. One interviewee stated that he had developed tools to automate critical project functions, such as testing and software documentation.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - (Re)Use existing Ada tools when possible. One interviewee recommended that public-domain toolsets such as STARS be used once the tool quality improves and the accessibility to the tools is improved. Another interviewee stated that he had found an Ada vendor that supplied a wealth of support software tools and provided a very powerful development environment. To accommodate these tools, he used this tool set for development, even the eventual target compiler and processor were different from the ones used in the development environment.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.19 ADA LANGUAGE COMPLEXITY

2.3.19.1 Problem Definition

The complexity of the Ada language makes it difficult to learn, use effectively, and test (validate). The Ada language requires extensive training for programmers to learn language syntax, proposed development methodologies, software engineering standards, and implementation issues for real-time embedded systems.

The Ada language is also difficult to use effectively and efficiently. For example, the current inefficiency of Ada compilers creates an environment where unchecked use of certain Ada features (tasking, generics, etc..) can cause significant impacts on system performance. Also, Ada development methodologies and programming standards/conventions are still being established.

The power and complexity of the Ada programming language can also make it difficult to test and validate an Ada system. The functional and performance impacts of the RTL, a very complex piece of software, play an important part in the testing process.

2.3.19.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Establish guidelines for use of the Ada language. The complexity of the Ada language and runtime environment requires that comprehensive guidelines be established for the effective use of Ada in a real-time embedded application. These guidelines can include Ada-oriented design, coding, and documentation standards as well as restrictions on the usage of certain Ada constructs.

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Perform extensive Ada training. The availability of comprehensive training support is critical to ensure that project personnel are properly trained in the effective use of the Ada language. The complexity of the language requires that training be provided in areas such as Ada programming, Ada real-time issues, the Ada runtime environment, and Ada "lessons learned".

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 3 - Obtain information concerning past Ada implementation efforts. Information obtained from other Ada developers can reduce the complexity of an Ada development effort by providing insight into the features of the language and the problems which can occur when developing Ada applications.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.19.3 Actual Industry Solutions

Solution 1 - Hire an experienced Ada staff. One interviewee stated that he had hired project personnel that had prior Ada and software development experience; they were better able to deal with the complexity of the Ada language. However, due to the current shortage of experienced Ada programmers, this solution is not always possible.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Create team of "Ada experts". One interviewee stated that he designated certain personnel as "Ada gurus"; they were responsible for resolving complex Ada issues. He attempted to maintain an acceptable ratio of gurus to developers. Another interviewee stated that he had created an "elite corps" of Ada software engineers to handle complex Ada issues; these engineers were separate from the software technicians (programmers) who implemented the Ada design.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.20 CUSTOMIZATION OF RUN-TIME LIBRARY

2.3.20.1 Problem Definition

When an Ada compiler and its respective runtime support code is validated, it is on a target system specified by the compiler developer. The problem is that an applications user of the compiler will usually have a different configuration (memory map, I/O map) for their particular system. Also, the user may desire different default values for their runtime application, such as the task stacks.

Therefore, the applications user must modify or customize the runtime support code. To get this type of customization the user will need to recompile the runtime support library. If the user desires to perform this task himself, he must usually purchase the source code rights for the RTL and train some people on how to perform the required task. The other option is to contract the vendor to make the required modifications. Either course of action will add more cost to the development of the application program.

Users have discovered on some Ada projects that the ALS runtime software needed to be reconfigured for memory map allocation, interrupt vector addresses, and Input/Output addresses for Intel's PIC and PIT chips. Further, RTL code had to be repackaged to allow for better selective linking and modified to remove unused 8087 code and to reduce interrupt overhead.

2.3.20.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Modify the RTL to meet project-specific and/or machine-dependent requirements. For some Ada applications, the performance requirements are so severe or specialized that it is necessary to modify or customize the RTL in order to meet these requirements. An example of this would be the use of a tasking implementation which performs a limited context switch. This customization can be performed by the compiler vendor or the developer (if the source code is available).

It should be noted that due to the complexity of the RTL, modification of this code should be performed as a last resort. Also, once this code has been modified in-house, it is the responsibility of the in-house personnel to maintain it from then on.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.20.3 Actual Industry Solutions

Solution 1 - Modify RTL to meet project functional and performance requirements. One interviewee stated that he modified the RTL in areas such as interrupt handling and configuration for target processor. Another interviewee stated that he had modified the RTL in areas such as task context switching, power-up initialization, and configuration for target processor.

A number of interviewees stated that they had performed their own removal of unnecessary routines from the runtime software environment. These interviewees also stated that they had requested that the compiler vendor provide a manual or automated capability to remove unnecessary software from the runtime environment.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Request that compiler vendor modify runtime software. One interviewee stated that he had requested that the compiler vendor modify the runtime software in areas such as interfacing to assembly language routines, task context switching, and power-up initialization.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.21 LACK OF EXPERIENCED ADA PROGRAMMERS

2.3.21.1 Problem Definition

As we in the Ada community have heard so often, Ada is "not just another high order language", but is a whole new approach to software design. Indeed, many respected individuals feel that the major contribution that Ada will make is to train programmers in the use of modern software design techniques [Ref]. When we speak of the problem of not enough experienced Ada programmers, it is in this broader context that we see the problem.

Recent experience shows that a fresh graduate with a CS degree can learn Ada syntax well enough in about three weeks to start making useful contributions to a project. Despite the complaints about Ada being "too complex", it turns out in practice that some of the complex features need only be used by a small percentage of design team members.

Teaching a methodology takes far longer. Formal courses typically go for two or three weeks at two or three hours per day (with homework), but it takes actual project experience before most people really understand the value of a methodology and become advocates of it. Without this kind of full support, most methodologies are reduced to being an extra paperwork burden in the minds of the programmers.

Formal courses are another problem because hiring is usually done over a period of time, rather than hiring the first few applicants who meet the minimum qualifications. Putting untrained people on the job while they are waiting for the next class to start brings on the infamous Brooks' Law effects, where the addition of additional programmers slows down the team [Ref]. This happens because the experienced people must spend considerable time explaining to the new people some of the things they would normally have learned in the classes.

Finally, the supply of experienced Ada programmers is not easy to measure. At the November 86 SIGAda Conference, for example, it was reported that the biggest shortage in Ada-trained people is among managers. The supply of programmers was greater than the number of people actually doing Ada work. However, this result flies in the face of some recent experience in hiring, where a large number designers have been hired to do Ada work and few of them had any Ada experience on the job.

This apparent discrepancy between the reported oversupply and the experienced shortage of Ada programmers could have been a local shortage or it could have been pure chance. But it also could have been caused by large firms training massive numbers

of people in Ada in anticipation of future Ada contracts. This would indeed cause an oversupply to be measured if one counted everyone who went through these courses as an experienced Ada programmer.

[Ref] F.P.Brooks, "No Silver Bullet", IEEE Computer, April 87

[Ref] F.P.Brooks, "The Mythical Man-Month", 1975, Addison-Wesley, Reading, Mass.

2.3.21.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Provide comprehensive training for project personnel. The current lack of experienced Ada software engineers and managers makes it critical that comprehensive training be provided to project personnel. This training helps to ensure that the Ada developers use the language effectively. It is important to note that this training should be provided to the software engineers, managers, and, if required, to the customer/client.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Provide comprehensive standards and guidelines for Ada software developers. The establishment of Ada standards and guidelines provides the Ada developers with information concerning the effective use of the Ada language and also provides guidance concerning the restrictions on the use of certain Ada constructs for an application.

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Establish team of Ada software "gurus" to support Ada development efforts. Due to the shortage of experienced Ada software engineers, the "gurus" are needed to provide design and implementation information to the project personnel. The establishment of this team can greatly improve project productivity by improving the effectiveness of the development team and minimizing the amount of rework performed on the project.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 4 - Automate the development environment as much as possible. The use of automated tools to perform many of the Ada development activities lessens the impact of the inexperienced Ada software engineers, while also serving to implement and enforce the standards established for the project.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 5 - Provide external consulting support, if required, for development staff. The use of an external consulting source to provide general and project-specific Ada information to the developers can be very beneficial to the project. This solution reduces the learning curve by providing Ada experience and expertise very early in the project when a number of critical development decisions must be made.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.21.3 Actual Industry Solutions

Solution 1 - Hire experienced Ada programmers. One interviewee stated that he had minimized this problem by hiring experienced Ada programmers. He acknowledged, however, that in many cases this is not a feasible solution due to the shortage of these programmers.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Hire experienced software engineers and train them in Ada. One interviewee stated that he had hired software engineers with good software development backgrounds and provided them with a combination of formal and on-the-job training.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Hire engineers with little Ada experience and provide them with comprehensive Ada training. One interviewee stated that he had used this approach because he was only able to find inexperienced Ada programmers and because he felt more comfortable with them once they had been through a formalized Ada training sequence.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.22 EXTENSIVE ADA TRAINING REQUIREMENTS

2.3.22.1 Problem Definition

The complexity of the Ada programming language and the system development and runtime environments results in extensive training requirements for Ada applications programmers.

To fully prepare the applications to develop Ada software a variety of training should be provided at different levels throughout the various phases of the project. Training should be provided at a minimum at the following levels - Senior Technical Staff, Junior Technical Staff, and Management.

The training courses to be offered should be selected according to the type and level of personnel being trained. The technical staff members should be offered all or some of the following courses:

- * Ada Language Overview
- * Advanced Ada Language Issues
- * Ada Development Methodologies
- * Ada Implementation Issues
- * Ada APSE Issues

The management staff should be offered the following courses:

- * Ada Language Overview
- * Ada Cost/Schedule Estimation and Tracking
- * Ada Development Environment
- * Ada Productivity Issues

Ada training costs are high and a large amount of time is required to train the programmers. From 3 - 12 weeks should be allotted for the technical staff and from 1-3 weeks for the management staff. The training courses should be scheduled to coincide with the proposed project staffing plan. It should be noted that it is more difficult to retrain non-Ada programmers than to train new programmers.

2.3.22.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Look for experienced software engineers. The availability of experienced Ada software engineers can reduce the extensive Ada training requirements for the project. Not only do these engineers require less training, but they can also provide training to less experienced personnel.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Customize Ada training needs for a specific project. Ada training needs should be determined early in the project, based on factors such as the type and complexity of the application, the experience and expertise of the project personnel, the status of the project, and the project training budget. Once the training needs have been determined, the project personnel can be provided with the training that is required for the particular project.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Provide internal training support ("train the trainers"). To reduce the training time and cost for Ada development efforts, one approach that can be used is to send the most experienced personnel to the training courses and then let them train the remaining project engineers.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 4 - Use automated tools to support training efforts. The use of automated self-paced training courses, as well as the availability of Ada software tools for hands-on training and practice, can reduce Ada training requirements.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.22.3 Actual Industry Solutions

Solution 1 - Train a team of Ada "gurus". One interviewee stated that he had trained a team of Ada "gurus" and had then used them to train other members of the development team.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Provide on-going training. One interviewee stated that he had used a combination of on-the-job training and limited instruction to train his Ada programmers. This instruction included method training.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.23 INACCURACY OF COST/SCHEDULE ESTIMATES FOR ADA PROGRAMS

2.3.23.1 Problem Definition

Most of the Ada problems recounted here cause cost/schedule perturbations for embedded mission-critical applications [Problems #1,3,4,5,6,7,8,9,10,11,13,14,15,16,17,18,19,20,21,22,23,24,25,26].

Because of the pervasiveness of the influence of almost every problem on cost and schedule performance, the task of estimating cost and schedule performance becomes even more complicated. Not only must you understand enough about software cost/schedule estimating, but you must also understand enough about the additional problems you get in Ada. This Ada problem knowledge is a very rare commodity today, which is one of the reasons this report is being written.

In an effort to help, there have been extensions to one of the most popular cost models, COCOMO [Ref], that attempt to account for some of the better-known Ada problems [Ref]. These include the instability of the compiler (major revisions every six months being the present norm) and the extra time it takes to train Ada programmers.

Even someone who has lived through the Ada problems might be hard-pressed to estimate their cost/schedule effects on a new project. The only real way to do this is to have someone in control of the project who understands the pitfalls well enough to avoid them, making their cost/schedule effects near zero (since avoidance costs are usually small).

But not all Ada-specific cost/schedule factors can be called problems. After all, the ultimate reason to use Ada is to save money, not to lose it. When a steady-state condition is reached after a few years of using Ada, developers should find the costs much improved. But again, since very few have reached this state, the actual gains are very difficult to estimate.

But even when all Ada-specific cost/schedule influences can be accounted for, which may be years in the future for many applications developers, the job of software cost/schedule estimating is anything but easy. This is a long-standing software engineering problem, with even estimation models with many years of good service being criticized for making errors of 100 percent or more [Ref].

In fact, some recommend collecting your own statistics for several project done using your own programming support environment rather than using the factors in the models [Ref]. Published model factors are based on data from hundreds of projects, but if your environment is significantly different than the industry norm then your responsiveness to these factors may indeed be unique.

[Ref] B.Boehm, "Software Engineering Economics", Prentice-Hall, Englewood Cliffs, NJ, 1981

[Ref] R.W. Jensen, "Projected Productivity Impact of Near-Term Ada Use in Software System Development, "Hughes Aircraft Co., Fullerton, CA 1985

[Ref] C.Kemerer, "An Empirical Validation of Software Cost Estimating Models", Communications of the ACM, May 1987

[Ref] H.Davis, "Measuring the Programmer's Productivity", Engineering Manager, February 85

2.3.23.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Use Ada-oriented software cost estimation tools. To ensure a more accurate estimation of Ada software development cost and schedule, a small number of Ada-oriented software cost estimation models are available (Ex: Revised COCOMO, Revised PRICE-S). These models have been revised to reflect the historical information that has been obtained from recent Ada projects, and should provide more accurate estimates than the older models.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Obtain historical data concerning estimated vs. actual cost/schedule performance on Ada projects. One way to improve the accuracy of cost/schedule estimates for Ada projects is to access the data base of recent Ada projects to obtain cost estimation information for similar projects. However, the information contained in this data base is very limited, due to the small number of Ada projects that have been completed.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 3 - Use non-Ada-oriented software cost estimation models and extrapolate for use on Ada projects. If the Ada development team does not have access to an Ada-oriented software cost estimation model, it may be feasible to extrapolate from a non-Ada-oriented model if there is a sufficient base of experience or historical data to ensure that the extrapolation is realistic.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 4 - Obtain personnel with experience in performing cost/schedule estimation for Ada projects. However, the availability of personnel with experience in performing software cost estimation for Ada projects is currently very limited.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 5 - Maintain historical data concerning estimated vs. actual cost/schedule performance on Ada projects. Current Ada developers should maintain information concerning their performance on Ada efforts so that this data can be used to improve the accuracy of the Ada software cost estimation process.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.23.3 Actual Industry Solutions

Solution 1 - Use cost estimation tools that have been updated for use on Ada projects. A number of interviewees stated that they have used or plan to use updated cost estimation tools such as the Revised COCOMO.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Hire experienced Ada software personnel. One interviewee stated that he had hired Ada software engineers with software cost estimation experience on Ada projects and that they had performed cost estimation for the project.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Attempt to use flexible contracting methods. One interviewee stated that due to the uncertainty of current Ada cost estimation techniques, he had performed his Ada efforts under a Cost-Plus contracting approach. Another interviewee stated that he performed his Ada efforts under a Time and Materials contract.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.24 LACK OF ESTABLISHED ADA SOFTWARE DEVELOPMENT METHOD

2.3.24.1 Problem Definition

One of the most important features of Ada is that it encourages the use of modern software engineering development practices [Problem #21]. These practices can best be exploited when they are packaged within a good development methodology. The lack of an established Ada development methodology can be a problem in that engineers trained in one methodology are less mobile if other organizations (even within the same parent organization) use a different one. This in turn contributes to the lack of experienced Ada programmers [Problem #21] since an important part of their experience is their methodology training.

This is not to say that there is a lack of good Ada methodologies. In fact there are two very good ones, Object Oriented Design (OOD) and PAMELA (TM George Cherry). One of these, OOD, is gaining both popularity and respect very rapidly, and PAMELA, while held back now by limitations described below, has good long-term potential also.

OOD is predicated upon the premise that problem definition is the first and hardest task a designer confronts. OOD concentrates on helping the designer with this task by stripping away much of the syntactic material that adds little to this task. It does this by defining the Object as an Ada package or task, which is a higher level view than the module of Yourdon [Ref]. This has led some to proclaim OOD as being the most promising of the "technical fads" now available to improve programmer productivity [Ref]. But it also causes some to complain that OOD is too limited when it comes to expressing the dynamism of Ada systems in operation [Ref].

PAMELA, on the other hand, is best at describing the dynamism of Ada systems. It is the first process-flow methodology specifically adapted to the Ada syntax [Ref 1], using easy to learn graphical techniques to input the design information that it will use to generate code automatically. This labor-saving step is also its biggest problem right now, however, since it generates an Ada task for every "single-thread" (no children) process [Ref]. With current Ada compilers the overhead due to context switching is too high to allow free use of tasking.

However, some vendors that have developed Ada applications using PAMELA have been forced to redesign extensively to cut down the number of tasks in the system because it could not meet the design constraints. This is the other objection to PAMELA: it leads to deeply-nested structures [Ref].

Note that neither of the two limitations to the use of PAMELA are necessarily long-lasting ones. When Ada compilers become available that can do very rapid context switches (perhaps in conjunction with underlying microprocessors that have richer

instruction sets) and that maintain a single task stack for the parent task and all direct descendants (to avoid context switches among them), PAMELA may become much more widely used.

As a final note, there are some software organizations that service customers outside the Department of Defense and thus may need to use languages other than Ada on some projects. happened to be "C." Since one of the main goals of a functional software organization is the balancing of labor resources between on-going and planned projects, the use of a non-language-specific methodology may be preferred. This allows programmers to be moved between major projects to satisfy special project needs or to further individual career goals.

[Ref] S.Boyd,"Ada Methods: Object-Oriented Design & PAMELA", SIGAda, Nov 86

[Ref] F.P.Brooks,"No Silver Bullet", IEEE Computer, Apr 87

2.3.24.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Select an Ada-oriented method for use as a standard. The selection of a method should be based the type of applications that are to be developed, and the method features that are desired ([Soni88]).

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Provide comprehensive guidelines for use of the selected method. To ensure effective use of the method, the Ada software engineers must be provided with comprehensive guidelines and standards for its use.

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Acquire automated tools to support and implement the selected method. The acquisition of automated tools improves project productivity by eliminating a number of manual development activities and also implements and enforces the established software development method standards and guidelines.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 4 - Provide comprehensive training in use of the selected method to ensure that the Ada development team understands how to use the method effectively. Proper method training is one of the most critical factors in the success of a project.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.24.3 Actual Industry Solutions

Solution 1 - Develop your own Ada-oriented method. One interviewee stated that he had developed his own method for use on Ada projects. This method was designed for use in real-time environments and was based on his experience on previous projects. This method is obviously not feasible in most cases due to the amount of software development experience that it requires to do it right.

Solution Method(s) - Methodology
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Modify an existing method for use on Ada project. One interviewee stated that he had modified an existing method for use in the development of real-time Ada applications. This modification had included the addition of Ada symbols and constructs, the use of Ada PDL, and the modification of documentation standards to reflect Ada issues.

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.25 LACK OF ESTABLISHED ADA STANDARDS AND GUIDELINES

2.3.25.1 Problem Definition

A growing number of software development organizations are measuring the productivity of each designer and manager, rewarding them primarily on this measured productivity and the quality of their software products. This has been proven over and over to be a sound practice since we have observed that the difference in productivity between the best and the worst programmers to be consistently about a factor of ten. Besides keeping turnover of the best people very low, this policy also strongly encourages the worst to either quickly improve or to find another line of work.

The reason all organizations don't use this practice is that it takes a considerable amount of effort to make it work right. The measurands we use are, by now, quite standard (operational lines of code and hours charged to the project), with good tools available to automate their collection. The hardest part is to establish standards that can be anchored in some facts, such as the ubiquitous "national average productivity", or a baseline formed from several similar projects done in our environment [Ref].

Here is the problem that Ada introduces. Because of the difficulties in estimating the number of lines of code or labor hours that a project should take [Problem #23], the credibility of the standard is jeopardized. As long as the standards were based on measured accomplishments of others, they were accepted as a challenge. But if they must be based on theory because Ada has a dearth of real data, their motivational value is greatly reduced.

Compounding the problem is the observation [Problem #23] that the productivity of Ada programmers is likely to start out worse than with other languages, then rapidly improve, ending up with higher productivity than other common languages. This makes productivity data collection far more difficult, since it is continually changing over a period of two or three years. It also makes the data from other organizations more suspect since it is hard to tell exactly where on the learning curve they might be operating.

[Ref] H.Davis, "Measuring the Programmer's Productivity", Engineering Manager, February 85

2.3.25.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Develop a set of Ada standards and guidelines and apply them consistently on Ada projects. The establishment of a comprehensive set of Ada software engineering standards is a means to ensure that project personnel use the Ada language effectively and that Ada constructs which are undesirable for a particular application are identified. The standards to be established can include design, coding, and documentation standards.

The development team should also evaluate the suitability of the standards for use on their project(s) and provide feedback to those responsible for implementing and enforcing the standards.

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Obtain information on standards and guidelines currently being used on Ada projects and use these, if possible. Existing Ada software engineering standards can provide a useful starting point for the development of project-specific standards and guidelines. It is also useful to obtain information concerning the experiences of the Ada software engineers when the standards were used on an actual project.

Solution Method(s) - Methodology
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 3 - Acquire automated tools to implement and enforce use of standards and guidelines. The use of automated tools will help ensure that the established standards are applied thoroughly and consistently across the project.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.25.3 Actual Industry Solutions

Solution 1 - Develop your own Ada standards and guidelines. One interviewee stated that that he had developed his own set of Ada software engineering standards and guidelines and provided them to the development team at the beginning of the project.

Solution Method(s) - Methodology
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Use existing Ada software engineering standards. One interviewee stated that he had used available Ada software engineering standards and modified them for use on his particular project.

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Provide guidelines for use of 2167A. One interviewee stated that they provided guidelines for their development staff in the use of DOD-STD-2167A to reduce the inconsistency of documentation developed on the project.

Solution Method(s) - Methodology
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.26 PRODUCTIVITY IMPACTS OF ADA

2.3.26.1 Problem Definition

Of the preceding 25 problems, 18 have been shown to influence the cost of an embedded Ada system. On the other hand it is often true that problem identification is the hardest part: once you know what the problem is, the solution is sometimes obvious. This report is therefore not as discouraging as it may seem at first reading.

In the case of Ada, there are some long-term cost benefits that are the ultimate reason it has been adopted as the standard language for Department of Defense embedded mission-critical software [Ref]. And other languages are not problem-free: MGEN Smith reported that 70 to 80 percent of the late Air Force projects are having software problems [Ref]. At the same conference MGEN Salisbury reported another need for a standard language like Ada that can handle a wide variety of applications: The Standard Army Management Information System (STAMIS) had grown to 750 systems, now reduced to 110. Using Ada is expected to cut it to 37 programs.

The one area which is probably the biggest source of Ada productivity problems is the speed of the compiler. Compilation speeds have been steadily dropping for VAX-host, Intel-target compilers. Some users have seen a times ten improvement here in the last two years, with another big improvement reported to be in the offing for a compiler to be validated in September 1987 [Ref].

For the present, however, Ada does have serious productivity impacts which can price it out of the market for many projects if you look at development costs only.

[Ref] Department of Defense Directive, 3405.2, Subject: Use of Ada in Weapon Systems, Mar 87

[Ref] Policy Committee Reports From Armed Services, SIG Ada, Nov 86

[Ref] L. Silverthorn, DDC-I, Phoenix, AZ

2.3.26.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Select Ada tools which provide good performance. One of the most important issues which affects project productivity is the availability and performance of automated software development tools. Currently, some of the available Ada tools (compiler, runtime environment, APSE, etc..) provide poor performance and thus reduce overall project productivity due to factors such as extensive optimization requirements.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Provide comprehensive Ada training. The availability of comprehensive Ada training can increase project productivity by reducing the overall learning curve and teaching the developers to use the Ada language effectively. Normally, for a medium to large project, these benefits more than offset the cost of the training.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.26.3 Actual Industry Solutions

Solution 1 - Limit the number of project variables. One interviewee stated that his approach to increasing the productivity of his development team was to limit the number of new development issues that must be addressed for the project. These issues include learning a new language (Ada), software development method, and documentation standards.

Solution Method(s) - Management
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Develop approach to measure productivity. One interviewee stated that he first defined what "productivity" was for his project. He then developed an approach, both manual and automated, to measure productivity. The results of these productivity measurements were then used to evaluate the impact of project decisions on productivity.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 3 - Use automated tools as much as possible. One interviewee stated that he developed automated tools to perform critical functions (such as testing) on his project. Another interviewee stated that he attempted to use existing automated tools as much as possible, even though he had to modify some of them for use on his project.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.27 IMPACT OF CONSTRAINT CHECKING ON SYSTEM PERFORMANCE

2.3.27.1 Problem Definition

Ada provides the capability, within the language, to perform constraint checking. Constraint checking provides the Ada application developer with a means to determine whether a variable was assigned a value at runtime that is outside of its defined range. When this event occurs, the exception `CONSTRAINT_ERROR` is raised to signal the problem. "However, the event that the requirements for constraint checking become too severe, Ada provides a `SUPPRESS` pragma to disable this feature."

2.3.27.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Use "`SUPPRESS`" pragma to disable constraint checking at runtime. In some Ada implementations, this pragma causes the range checking code to be removed from the runtime environment. In other implementations, the pragma causes the range checking code to be bypassed at runtime; however, the code remains resident in the target environment.

It is also possible to selectively use the "`SUPPRESS`" pragma to eliminate range checking in specific portions of the application while continuing to perform it in other portions.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Remove constraint checking software from RTL. If the range checking feature of the compiler will never be used during system operation, the code can be removed from the runtime environment altogether (not generated by the compiler) once the development phase is complete. This modification can be performed by the compiler vendor.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.27.3 Actual Industry Solutions

Solution 1 - Turn off constraint checking. One interviewee stated that he turned off constraint checking once the development phase had been completed to improve system performance. He also performed constraint checking within the applications code to reduce the need for constraint checking to be performed by the RTS.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.28 INABILITY TO ASSIGN DYNAMIC TASK PRIORITIES

2.3.28.1 Problem Definition

Ada does support a capability for dynamically altering the priority of a currently running task. The value for the pragma `PRIORITY` is static and therefore cannot be changed at runtime. Implementations may support an alternate set of priorities that control tasking in the case where the Ada `PRIORITY` is identical or undefined. This allows an implementation-defined subpriority, which may be dynamic, to control the scheduling. This capability is not supported by many implementations, and a standard does not exist to help provide commonality.

2.3.28.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Use of Ada subpriorities (if available). The Ada subpriority feature can be used to provide the Ada task scheduler with another piece of information to be used in determining the order in which Ada tasks will be activated. However, this approach is not an adequate substitute for the ability to dynamically change the priority of a task.

Solution Method(s)	- Technical
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Modify Ada compiler to allow use of dynamic task priorities. This feature is currently being considered for inclusion into the Ada compiler, although it currently violates MIL-STD-1815A which states that priorities must be static values.

Solution Method(s)	- Tools
Solution Timeframe	- Long-term
Solution Approach	- Preventive

2.3.28.3 Actual Industry Solutions

Solution 1 - Obtain an Ada compiler that provides runtime support for the use of dynamic priorities. One interviewee stated that his project had used a compiler which provided the capability to dynamically change the priority of a task.

Solution Method(s)	- Tools
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Modify runtime software to provide dynamic priorities. One interviewee recommended that the compiler vendors could provide access to their runtime software so that the Ada developers could implement this function, if desired.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.29 INABILITY TO PERFORM PARALLEL PROCESSING

2.3.29.1 Problem Definition

Current implementations of Ada do not support true parallel processing, which involves distributing the processing to be performed among a number of processors and supporting the sharing of information concerning system programs and data between the various processors. For current multiprocessor applications that are implemented in Ada, an independent and stand-alone version of the RTS must reside on each processor and no facility is provided for communication between RTSs.

2.3.29.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Use of a distributed RTL (when available). A distributed RTL will provide Ada developers with the capability to perform true parallel processing operations such as executing a single process which has subsets of its functions running concurrently on multiple processors.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Use of special-purpose hardware and software to implement parallel processing. In some cases, the requirement for parallel processing can be implemented through the use of hardware that can perform operations independent of and concurrently with the target processor. This hardware could consist of a numeric co-processor or an array processor.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.29.3 Actual Industry Solutions

Solution 1 - Develop your own RTL that supports true parallel processing. One interviewee stated that he had done this and had also modified his applications code to use the parallel processing capabilities. This can be a very difficult task due to the complexities of the RTL.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 -Request that the Ada compiler vendors develop run-time systems that support true parallel processing. One interviewee recommended that the compiler vendors develop parallel processing systems; however, he acknowledged that these systems will not be available for some time.

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.30 LACK OF SUPPORT FOR LOW LEVEL OPERATIONS

2.3.30.1 Problem Definition

"Ada does not provide a mechanism to control the processor state (including interrupt masks required for critical sections). Although Ada provides a mechanism to directly manipulate memory mapped hardware, no capability exists within the language to access internal processor registers. Such a mechanism would be difficult to standardize."

For example, "...changing the processor state needs to be done in conjunction with the runtime. Since stacks used for different states are often separate, simply changing state will result in an error condition. Also, subsequent calls to the runtime (possibly due to exceptions) are likely to cause unpredictable results."

Another example is that "...there is frequently a need to enable and disable interrupts which is performed by setting or clearing interrupt masks. It is easy for a programmer to write an assembly language routine to manipulate an interrupt mask and call this routine from an Ada program. The problem occurs because the assembly language is not working in conjunction with the runtime environment provided."

2.3.30.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Develop low-level support software in Ada. By developing the low-level support routines in Ada, the overall software system becomes more maintainable and the low-level routines can be more easily integrated into the Ada application. The implementation of the low-level software can be hidden from the rest of the application by concealing the implementation in a package or by writing the code in another language and interfacing this language to Ada.

Solution Method(s)	- Technical
Solution Timeframe	- Short-term
Solution Approach	- Remedial

AD-A223 262

REAL-TIME ADA PROBLEM SOLUTION STUDY(U) SONICRAFT INC
CHICAGO IL S J MCCULLOUGH ET AL. 24 MAR 89

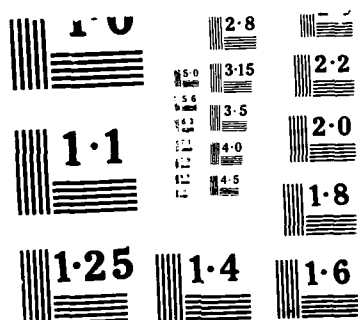
2/2

UNCLASSIFIED

F/G 12/5

NL

END
FILMED
DTIC



Solution 2 - Use existing non-Ada software routines to performing low-level operations. If these routines can meet the system performance requirements, then it may be feasible to make external calls to these routines from the Ada application. However, it should be noted that it is not always easy or possible to interface Ada programs with external non-Ada programs; it depends on the implementation language for the non-Ada routines.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.30.3 Actual Industry Solutions

Solution 1 - Minimize use of low-level operations. One interviewee stated that he minimized the use of low-level operations through the use of assembly language or external software (via Pragma Interface) to perform the low-level functions.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 2 - Use in-line code statements. One interviewee stated that he used in-line code statements to perform low-level operations in conjunction with the RTS.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Modify run-time software. One interviewee recommended that the run-time software could be modified by the applications developers to perform low-level operations such as providing access to hardware registers and obtaining more control over interrupts.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.31 INABILITY TO PERFORM TASK RESTART

2.3.31.1 Problem Definition

Applications which require that a separate thread of control (task) be restarted at the beginning after being interrupted part way through have difficulty mapping this requirement to Ada.

"Certain applications do have a need to be able to have multiple tasks, where one task might be pre-empted by a higher priority task, and the result of the pre-emption is to make the continuation of the pre-empted task meaningless."

"The standard Ada solution to this problem is to ABORT the pre-empted task, and then re-activate a new task. This creates a few undesirable side effects, not the least of which is likely to be unacceptable performance degradation."

2.3.31.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Minimize or eliminate the use of Ada task abort and restart mechanisms. The use of the task abort and restart mechanisms can cause unpredictable and sometimes severe side effects. One side effect is performance degradation due to the overhead associated with task activation and deactivation. The task abort mechanism is non-deterministic and involves the deactivation of the task and all of its dependent tasks. These side effects can affect overall system reliability.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Modify Ada RTL to improve the task restart process. Currently, Ada developers are considering ways to improve the reliability of the task abort/restart process.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Remedial

2.3.31.3 Actual Industry Solutions

Solution 1 - Provide low-level tasking support. One interviewee recommended that the compiler vendor could supply a "low-level tasking" package that would provide the developer with an interface to the RTS that would allow more control of the tasking process.

Solution Method(s)	-	Technical
Solution Timeframe	-	Long-term
Solution Approach	-	Remedial

2.3.32 INABILITY TO PERFORM CYCLIC SCHEDULING IN ADA

2.3.32.1 Problem Definition

Cyclic scheduling provides the capability to perform periodic processing by running a number of processes on a scheduled time basis. "The Ada language can support some degree of periodic processing by using the DELAY statement. Although some implementations provide a reasonable mechanism for this, the DELAY statement is not always adequate for this application."

"The problem [with the DELAY statement] is that the duration value is a delay from the current time, not a fixed interval. Therefore, the clock must be read and the cycle computed in the simple_expression allowed for the DELAY statement. However, there is no way to ensure that an interrupt (and possibly a higher priority task) is not executed between the time the clock is read in the simple_expression and when the delay duration is actually interpreted by the runtime [program]."

2.3.32.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Use knowledge of Ada scheduling algorithm to increase system determinism. This solution attempts to perform cyclic scheduling by making use of knowledge of the task scheduling algorithm used in a particular Ada implementation. This approach has two major problems. The first is that the scheduling algorithm can differ from Ada implementation to Ada implementation. The second is that the DELAY statement, which is used to implement cyclic scheduling is also non-deterministic.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Minimize use of cyclic scheduling in the system design. This eliminates the need to address the problems of non-determinism that are associated with cyclic scheduling in Ada.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

2.3.32.3 Actual Industry Solutions

Solution 1 - Use an external device to perform cyclic scheduling. One interviewee stated that he used an external device to provide interrupts which defined operating periods (replacing delay statements). These interrupts were used to implement cyclic scheduling for the Ada application.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Request that compiler vendors provide a package that performs cyclic scheduling functions. One interviewee recommended that the compiler vendors could provide this capability, perhaps by using a table-driven scheduler.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Remedial

2.3.33 LACK OF FLOATING POINT COPROCESSOR SUPPORT

2.3.33.1 Problem Definition

"A floating point coprocessor is a high performance numerics processing element that extends the main processor architecture by adding significant numeric capabilities and direct support for floating point, extended integer, and BCD data types. The presence of a floating point chip would increase performance in a real-time embedded application that required floating point operations to be performed."

"There is a lack of a standard for floating point coprocessor support in Ada. Some compilers require a floating point chip to perform floating point processing; other compilers cannot utilize the chip if it is present."

2.3.33.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Acquire an Ada compiler which provides support for a floating point co-processor. A number of the current Ada compilers support the use of the co-processor if it is present in the system implementation and perform software floating point arithmetic if it is not.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Modify compiler to provide support for floating point co-processor. If the compiler does not support the use of the floating point co-processor, the compiler vendor can perform this modification.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 3 - Develop software to utilize floating point co-processor. The Ada developer can write software to utilize the floating point co-processor and integrate it into the applications program or into a utilities library.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.33.3 Actual Industry Solutions

Solution 1 - Obtain an Ada compiler which provides coprocessor support. One interviewee stated that he had used a compiler which provided a switch for the use of a coprocessor.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Minimize the use of floating point computations. One interviewee stated that he did not use floating point computations in his Ada application. This may not be feasible for applications that must perform heavy scientific or signal processing.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Standardize the floating point operations. One interviewee recommended that the compiler vendors attempt to standardize on a format, such as the IEEE format, for handling coprocessor data (floating point objects).

Solution Method(s) - Management
Solution Timeframe - Long-term
Solution Approach - Remedial

2.3.34 INABILITY TO RECOVER FROM CPU FAULTS IN ADA

2.3.34.1 Problem Definition

"CPU fault tolerance is the built-in capability of a system to provide continued correct execution in the presence of a limited number of hardware or software faults. Highly reliable systems require that the software continue to operate in the presence of CPU faults."

"Although this may seem impossible, careful analysis indicates that many faults are momentary and do not result in permanent interruption of processing capability. However, it is essential that the program be able to recover from such faults and continue execution from the last check point. Ada does not directly support the ability to recover from such CPU faults."

2.3.34.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Modify RTL to provide support for CPU fault recovery. The RTL would have to be modified so that in the event of a CPU fault, control would be passed to the software that is responsible for identifying the fault and instituting recovery procedures. Once these activities have been completed, the fault recovery software returns control of the processor to the RTS.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.34.3 Actual Industry Solutions

Solution 1 - Modify run-time software to handle CPU faults. One interviewee stated that he had written assembly language routines to be inserted into the run-time environment for handling CPU faults.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Obtain an Ada compiler that provides capability to handle CPU faults. One interviewee stated that he had obtained an Ada compiler that provided the capability to tie interrupts to certain CPU faults (by modifying fault vectors).

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.35 IMPACT OF ADA COMPILER VALIDATION ISSUES

2.3.35.1 Problem Definition

"Validation is the process of checking the conformity of an Ada compiler to the Ada programming language [as specified in MIL-STD-1815A] and of issuing certificates indicating compliance of those compilers that have been successfully tested. It should be emphasized that the intent is only to measure conformance with the standard. Any validated compiler may still have bugs and poor performance, since performance is not being measured by the validation tests."

"To obtain a validation certificate, a compiler implementer must exercise an Ada Compiler Validation Capability (ACVC) test suite. The current level is Version 1.9 and it contains a series of over 2500 tests designed to check a compiler's conformance to the DoD's Ada language standard, ANSI MIL-STD-1815A (1983)."

"With the initial validation phase completed for most compilers, the compiler implementers are [finally] shifting their emphasis to concentrate on improving the efficiency of the generated code (code optimization) and providing more user configurability of the runtime environment."

2.3.35.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Evaluate need for project-validation of Ada compiler. If the project-specific requirements for compiler performance are critical enough to require modifications to the compiler, the impact of the project-validation process must be considered. This includes (re)certification of the project-validated status and maintenance of the modified compiler.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

Solution 2 - Ensure potential and plans for revalidation of Ada compiler. Since the Ada compiler must currently be revalidated every year, the compiler vendor's plans to support the revalidation process must be considered during the compiler selection process.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Preventive

Solution 3 - Evaluate results of proposed ACVC performance tests (when available). The Ada compiler validation team is currently considering adding a number of performance tests to the compiler validation process. The addition of these could have a significant positive impact on the quality and performance of available Ada compilers.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.35.3 Actual Industry Solutions

Solution 1 - Provide means during Ada compiler validation process to perform additional tests. A number of interviewees stated that the proposed ACEC tests would provide additional indications of compiler performance.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

2.3.36 INABILITY TO PERFORM ASYNCHRONOUS TASK

2.3.36.1 Problem Definition

"The Ada rendezvous model uses a synchronous mechanism to communicate between tasks. Many applications require that a signaling task not be delayed until the signaled task is ready to accept the signal. The mechanism used to communicate between tasks in the Ada rendezvous model is that both tasks must be synchronized together before any data or control information can be transferred."

"The Ada solution to this issue is to place an intermediate task between the signaling task and the waiting task. This intermediate task would always be ready for a rendezvous and would effectively buffer the transaction to provide asynchronous communications. The impact is to create an additional (logical) context switch."

2.3.36.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - Place an intermediate task between the signaling task and the waiting task. The use of an intermediate task to wait for a rendezvous and provide a buffer for the transaction has become the classical Ada solution to this problem. However, this solution has a drawback; the effect of using the intermediate task is to create an additional context switch, which increases system overhead.

Solution Method(s)	- Technical
Solution Timeframe	- Short-term
Solution Approach	- Remedial

2.3.36.3 Actual Industry Solutions

Solution 1 - Use buffering mechanism. A number of interviewees stated that they used a buffer task to implement asynchronous communications between tasks.

Solution Method(s)	- Technical
Solution Timeframe	- Short-term
Solution Approach	- Remedial

Solution 2 - Obtain an Ada compiler which provides run-time support for asynchronous tasking. One interviewee stated that he used an Ada compiler which provided non-Ada support for asynchronous tasking through the use of simple "signal and wait" operations.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.37 LACK OF IMPLEMENTATION OF CHAPTER 13 FEATURES

2.3.37.1 Problem Definition

"Many of the features in Chapter 13 [of the Ada Reference Manual] are not implemented in current commercially available compilers today. Chapter 13 of the Reference Manual for the Ada Programming Language is titled, "Representation Clauses and Implementation-Dependent Features". These features are optional and therefore a compiler can have the status of "validated" without any of these features implemented. However, many people feel that Chapter 13 is required for real-time embedded applications."

The features addressed in Chapter 13 of the Ada Reference Manual allow an Ada application developer to perform systems programming tasks by providing a physical representation of the underlying machine. These features include:

- * Representation Clauses
- * Length Clauses
- * Enumeration Representation Clauses
- * Record Representation Clauses
- * Address Clauses
- * Address Clauses For Interrupts
- * Change Of Representation
- * The Package SYSTEM
- * System-Dependent Named Numbers
- * Representation Attributes
- * Representation Attributes Of Real Types
- * Machine Code Insertions
- * Interface To Other Languages

2.3.37.2 Theoretical Solutions

The following theoretical solutions are proposed for the generic Ada problem:

Solution 1 - The Ada compiler vendor can implement desired Chapter 13 features. The implementation of Chapter 13 features is "optional"; a compiler can be validated without having any of these features implemented. However, a number of Ada developers feel that these features are necessary for real-time embedded applications. The Ada developer can request that the compiler vendor implement some or all of these features, depending on which ones were supplied originally with the compiler.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Ada applications developers can develop their own implementation of Chapter 13 features. If desired, the Ada developer can implement some or all of the Chapter 13 features, although there could be some difficulty integrating these features into an existing run-time environment.

Solution Method(s) - Technical
Solution Timeframe - Short-term
Solution Approach - Remedial

2.3.37.3 Actual Industry Solutions

Solution 1 - Obtain a compiler which provides support for some of the Chapter 13 features. One interviewee stated that he obtained a compiler which supported most of the Chapter 13 features. He developed software for his application which performed those desired Chapter 13 functions that were not provided by the compiler.

Solution Method(s) - Tools
Solution Timeframe - Short-term
Solution Approach - Remedial

Solution 2 - Provide additional ACVC tests that check for Chapter 13 features. One interviewee stated that the newest suite of ACVC tests contains a number of additional tests to check for Chapter 13 features.

Solution Method(s) - Tools
Solution Timeframe - Long-term
Solution Approach - Preventive

3. ANALYSIS AND CONCLUSIONS

3.1 Analysis

The following is a summary of the analysis of the solutions obtained during the Ada Problem/Solution Study.

1) The majority of actual solutions proposed for the performance-related generic Ada problems were resolved by revising the system design and through the use of tools, were short-term solutions, and were remedial in nature.

2) Most of the solutions proposed for the management-related generic Ada problems were resolved via management means or through the use of methods, were short-term and preventive in nature.

3) Most of the actual solutions proposed for the tool-related generic Ada problems were resolved by modifying the tools or acquiring new tools, were short-term solutions and were remedial in nature.

4) Overall, the large majority of the solutions were short-term in nature.

3.2 Conclusions

The following is a summary of the conclusions obtained as a result of the Ada Problem/Solution Study:

1) The short-term, remedial nature of the solutions proposed for the performance-related generic Ada problems reflects the general inexperience of the Ada community in developing real-time embedded Ada applications. This inexperience causes the developers to address the generic Ada problems in a "crisis" fashion; that is, the problem is not considered until it actually occurs.

2) The short-term, preventive nature of the management solutions to the generic Ada problems reflects a growing understanding of the up-front investment and planning that are required to successfully develop Ada applications. This up-front investment include training and acquisition of Ada tools. However, Ada developers still are not performing enough long-term planning to support their Ada projects.

3) The short-term, preventive nature of the tool-related generic Ada problems reflects a growing understanding of the importance of automated tools to support the development of Ada applications. However, it also reflects a lack of long-range planning to acquire the variety of tools that are required to develop complex Ada applications.

4) The overall short-term nature of the proposed solutions reflects a strong need for additional education in the areas of Ada software development planning and strategy.

4. RESULTS AND RECOMMENDATIONS

4.1 Results

The following is a summary of the results that were obtained during the Ada Problem/Solution Study.

- 1) A number of companies are enlisting the aid of compiler tool vendors to assist in solving functional and performance problems that stem from the Ada compiler and RTL.
- 2) Companies are attempting to identify potential problems early in the development process by assembling teams of "Ada gurus" to provide technical support to the development team.
- 3) Compiler improvements are reducing the need for the use of alternate methods (redesign, optimization, tool modification, etc..) to improve system performance for Ada applications. These improvements include more efficient code generation, specialized pragmas, and others.
- 4) Ada software tool vendors are building a larger number and variety of Ada tools for use in developing Ada applications. These tools are providing increased productivity and performance for Ada development efforts.
- 5) Ada developers are spending more time up front evaluating Ada tools to determine their suitability for use on a particular application. Better benchmarking techniques and data are becoming available for analyzing the performance of Ada tools.
- 6) Ada developers are reducing performance problems by providing guidelines for the use of certain Ada constructs which can adversely impact system performance. These guidelines are dependent on the characteristics of the application that is being developed and the Ada compiler that is being used.
- 7) Ada developers are improving system performance and functionality by performing their own modifications to the Ada RTL. These improvements include tailoring and/or configuring the RTL, writing additional code to perform required functions, and rewriting portions of the RTL.

8) Ada developers are still writing portions of their applications in assembly language to improve system efficiency and performance. This practice is expected to continue until the code generated by Ada compilers improves significantly.

9) Developers are designing system architectures for their applications which reflect the characteristics of the Ada applications that are being built. These architectures attempt to implement certain Ada features, such as context switching, in a more efficient manner.

10) Developers are providing more comprehensive Ada training to their employees. This training includes areas such as use of the Ada language, Ada software development methodologies, Ada software project management, Ada "lessons learned", and real-time Ada programming techniques. The availability of more comprehensive Ada training courses is increasing the productivity of the Ada software development team.

11) The proposed revisions to the Ada compiler definition, to be known as Ada 9X, are expected to resolve some of the problems currently facing Ada developers. These revisions are based on the feedback that has been provided by the Ada community during the first phase of Ada usage.

12) Ada standards and guidelines are now available and are being used by the Ada community. These standards are being refined as feedback is obtained from the Ada users. The availability of these standards is providing needed guidance to some of the less experienced Ada developers.

Figure 3, entitled "Problem Solutions Results", provides a summary of the study results. This figure describes the solution methods used to resolve the generic Ada problems, and the solution timeframes and approaches for both the theoretical and actual problem solutions.

4.2 Recommendations

The following is a summary of the recommendations that were made as a result of the Ada Problem/Solution Study.

1) Ada developers must obtain extensive education and training in the areas of Ada software development strategy, planning, project management, and problem identification/resolution. This can be obtained through formal training or informally from the experiences of other Ada developers.

PROBLEM SOLUTIONS (THEORETICAL VS. ACTUAL)

<u>SOLUTION METHOD</u>	<u>THEORETICAL</u>	<u>ACTUAL</u>
TECHNICAL	25%	30%
TOOLS	50%	43%
MANAGEMENT	16%	16%
METHODOLOGIES	9%	11%

FIGURE 3: PROBLEM SOLUTIONS RESULTS

PROBLEM SOLUTIONS (THEORETICAL VS. ACTUAL)

<u>SOLUTION TIMEFRAME</u>	<u>THEORETICAL</u>	<u>ACTUAL</u>
SHORT - TERM	83%	82%
LONG - TERM	17%	18%
<u>SOLUTION APPROACH</u>	<u>THEORETICAL</u>	<u>ACTUAL</u>
PREVENTIVE	60%	51%
REMEDIAL	40%	49%

FIGURE 3 (CONT.): PROBLEM SOLUTIONS RESULTS

PROBLEM SOLUTIONS BY TIMEFRAME **(THEORETICAL)**

<u>SOLUTION METHOD</u>	<u>SHORT-TERM</u> (83%)	<u>LONG-TERM</u> (17%)
TECHNICAL (25%)	21%	4%
TOOLS (50%)	42%	8%
MANAGEMENT (16%)	13%	3%
METHODOLOGIES (9%)	7%	2%

FIGURE 3 (CONT.): PROBLEM SOLUTIONS RESULTS

PROBLEM SOLUTIONS BY APPROACH **(THEORETICAL)**

<u>SOLUTION METHOD</u>	<u>PREVENTIVE</u> (60%)	<u>REMEDIAL</u> (40%)
TECHNICAL (25%)	15%	10%
TOOLS (50%)	30%	20%
MANAGEMENT (16%)	10%	6%
METHODOLOGIES (9%)	5%	4%

FIGURE 3 (CONT.): PROBLEM SOLUTIONS RESULTS

PROBLEM SOLUTIONS BY TIMEFRAME **(ACTUAL)**

<u>SOLUTION METHOD</u>	<u>SHORT-TERM</u> (82%)	<u>LONG-TERM</u> (18%)
TECHNICAL (30%)	25%	5%
TOOLS (43%)	35%	8%
MANAGEMENT (16%)	13%	3%
METHODOLOGIES (11%)	9%	2%

FIGURE 3 (CONT.): PROBLEM SOLUTIONS RESULTS

PROBLEM SOLUTIONS BY APPROACH (ACTUAL)

<u>SOLUTION METHOD</u>	<u>PREVENTIVE</u> (51%)	<u>REMEDIAL</u> (49%)
TECHNICAL (30%)	15%	15%
TOOLS (43%)	22%	21%
MANAGEMENT (16%)	8%	8%
METHODOLOGIES (11%)	6%	5%

FIGURE 3 (CONT.): PROBLEM SOLUTIONS RESULTS

2) There is a need for guidelines among the members of the Ada community, to include developers and tool vendors, and customers (such as the Government). However, these guidelines should be flexible enough to allow experimentation among the members of the Ada community, possibly leading to the establishment of industry and language standards.

3) The Ada user community must continue to put pressure on the Ada compiler and tool vendors to produce Ada tools that have much improved functionality and performance as well as higher reliability. This is very important in establishing the credibility of Ada for use in the development of real-time embedded applications.

4) The Ada community must be willing to invest their own resources (money, time, personnel, tools, training) to improve the current Ada development environment. It is not enough to expect the Government to fund all of the necessary investment in Ada technologies. This is becoming increasingly important with the entry of Ada into the commercial sector.

APPENDIX A: DEFINITIONS

Ada-oriented: The ability of a method to map the software design directly into the use of Ada language constructs such as packages, tasks, and generics.

Approach: A way of beginning or managing an effort; a way of analyzing, planning, or directing a project; a way of conducting operations. A scheme is an approach when it suggests ways to identify goals initially and/or suggests, at an abstract level, ways to proceed without goals. [Tele 87]

Auto Coding: Auto coding represents the capability, using a software tool, to automatically generate Ada source code statements. The tool user must provide specification information such as data definitions, module names, call decisions, etc.. [PJAC87]

Data Visibility: The extent to which a method provides visibility into the data that is used within an Ada software system. This visibility includes such issues as data flows, data definitions, and static vs. dynamic data.

Design Consistency: The extent to which an Ada software development method provides guidelines for consistent application of the method's principles. The benefit of this feature is that it encourages the development of a consistent software design among the different parts and participants of a software project.

Design Quality: The extent to which a method provides guidelines for determining and evaluating the quality of Ada software designs. This feature also provides a measure of the quality of the software that is designed using a method (reliability, maintainability, testability, portability, correctness, efficiency, understandability, etc..)

Ease of Learning: The ease with which an Ada-oriented method can be learned. This includes factors such as the amount of training that is recommended (based on the complexity of a method), the levels of training that are recommended (beginner, intermediate, and advanced), and the amount of time that is required before a software engineer can effectively use a method.

Ease of Use: The ease with a method can be used to design and implement an Ada software system. Ease of use is based on the simplicity, completeness, and consistency of the underlying method's principles, terminology, symbology, and products (documentation and deliverables).

Efficiency: The extent to which a software component fulfills its purpose with minimum use of computer resources.

Information Hiding: The extent to which a method supports information hiding for Ada software systems (such as through the use of packages). The principle of information hiding suggests that modules should be specified and designed so that information (procedure and data) contained within a module is inaccessible to other modules that have no need for such information. [SEPA]

Method: A definite, established, logical, or systematic plan. The steps and purposes have been thought beforehand in detail. A scheme is a method when it guides the user to a predictable result, given an appropriate set of starting conditions. [Tele 87]

Methodology: The study of methods. [Tele 87]

Methodology Tools: A measure of the availability and maturity of automated software tools which implement an Ada-oriented software method. The issue of availability involves the variety of tools that exist, the hardware/software environments in which these tools run, and the interface between these tools and the APSE.

PAMELA: Process Abstraction Method for Embedded Large Applications, a trademark of George Cherry for his Ada design method.

Portability: The ease with which a software component can be made functional in a different application or target computer architecture.

Problem Definition: The extent to which a method allows a software developer to define and represent a problem and its proposed solution during the development of an Ada software system. This is based on the completeness of the underlying principles, terminology, and symbology of a method. It is a measure of the ability of a method to model and represent the real-world.

Process State: The extent to which a method provides information concerning the state of the various processes which make up an Ada software system. This includes guidelines to establish state changes for Ada design efforts and guidelines for using symbology to represent state changes.

Process Visibility: The extent to which a method provides visibility into the functions and interfaces of the processes within an Ada software system. This visibility includes such issues as process control flow, concurrency (tasks), and external control (interrupts).

Program and Data Structure: The extent to which a method provides guidelines for using Ada language features which impose program and data structure. The program/data structure can take a number of forms, to include a flat hierarchy or a layered hierarchy.

Real-Time Ada: A computer program written in the Ada language which implements one or more real-time functions, usually triggered by interrupts.

Real-Time Function: Any system function (hardware, software or a combination) which is considered to have faulted if it has not been completed within a specified time after a signal to start.

Reusability: The ease with which a software component can be used or modified for use in another application.

Runtime Environment (RTE): The RTE consists of three functional areas: abstract data structures, code sequences, and predefined subroutines. It includes all of the runtime support routines, the conventions between the runtime routines and the compiler, and the underlying virtual machine of the target computer. "Virtual" is used in the sense that it may be a machine with layered software (a host operating system). An RTE does not include the application itself, but includes everything the application can interact with. In the event that there isn't any operating system layer (the bare machine target), the RTE includes those low-level functions found in an operating system.

Runtime Library (RTL): The RTL is a library of procedures and functions from which the RTS routines are selected.

Runtime System (RTS): The RTS is the set of subprograms which may be invoked by linking, loading, and executing object code generated by an Ada compiler. If these subprograms use or depend upon the services of an operating system, then the target runtime system includes those portions of that operating system. These predefined subroutines are chosen from the RTL for that Ada compilation system.

Task: Any program unit which is designed to be able to operate in parallel with other program units and to synchronize with them where necessary.

Traceability: The extent to which an Ada-oriented software development method provides the ability to validate the products of the various steps of a method and to verify that the input to each step fulfills the requirements levied by the previous step.